Wstęp

Książka obejmuje ćwiczenia z programu Borland Delphi 7. Skorzystać z niej mogą również posiadacze wcześniejszych wersji Delphi, jednak wersje te nie mogą być niższe niż 3.0.

Książka przeznaczona jest dla początkujących użytkowników, znających obsługę systemu Windows. Pozwala ona na szybkie, w miarę przyjemne, rozpoczęcie nauki programowania w zintegrowanym środowisku programistycznym Delphi.

Borland Delphi jest środowiskiem programistycznym dającym możliwość obiektowego i wizualnego projektowania aplikacji, przez co zaliczany jest do narzędzi typu RAD (Rapid Application Development) – co oznacza szybkie tworzenie aplikacji.

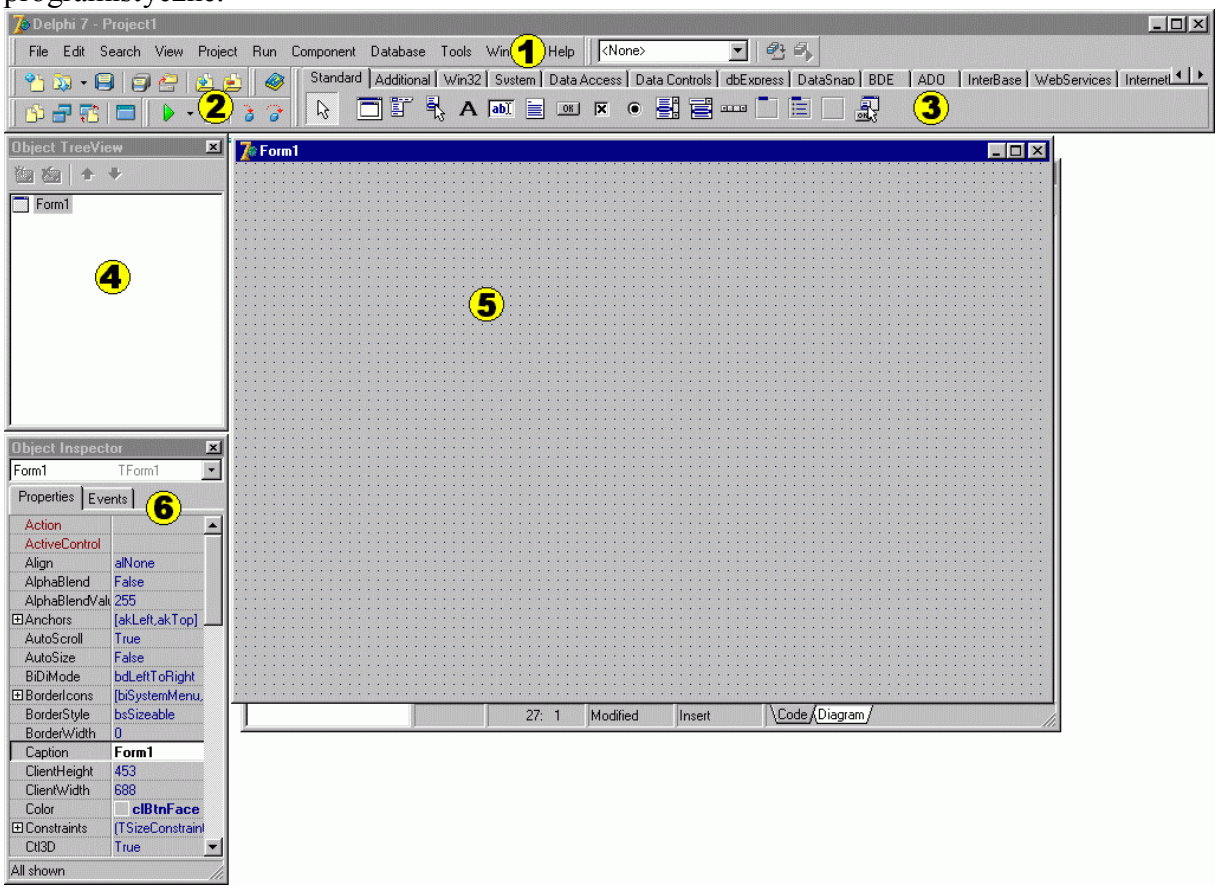
Niektóre przedstawione ćwiczenia, znajdują się na dołączonej do książki dyskietce. Jeżeli dane ćwiczenie znajduje się na dyskietce, to czytelnik zostanie o tym poinformowany. Znajduje się tam również kilka nowych komponentów, które można zainstalować.

Mam nadzieję, że książka ta przyczyni się do szybkiego poznania Delphi i tworzenia programów za pomocą tego języka.

Autor

1. Środowisko programistyczne

Po uruchomieniu Delphi – rysunek 1.1 zobaczymy zintegrowane środowisko programistyczne.



Rysunek 1.1. Widok środowiska programistycznego Delphi 7

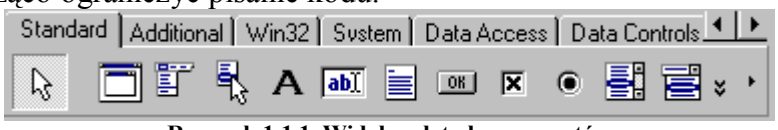
Środowisko to składa się z następujących elementów:

1) Menu;
2) Paska narzędzi – znajdują się tam klawisze z najważniejszymi funkcjami programu;
3) Paleta komponentów;
4) Object TreeView (Podgląd drzewa obiektów) – okno to służy do wyświetlania struktury umieszczonych obiektów na formularzu;
5) Formatka (domyślna nazwa Form1) – na niej umieszczane są komponenty w trakcie tworzenia programu;
6) Object Inspector (Inspektor obiektów) – w tym oknie umieszczona jest lista właściwości i zdarzeń. Lista ta zmienia się w zależności od tego, jaki obiekt zostanie wybrany (formatka lub komponent).

1.1 Paleta komponentów

Komponent jest to fragment kodu, który można wykorzystywać w czasie tworzenia programu. Komponenty reprezentowane są przez rysunki, które mają obrazować ich przeznaczenie. Służą do budowy funkcjonującego programu.

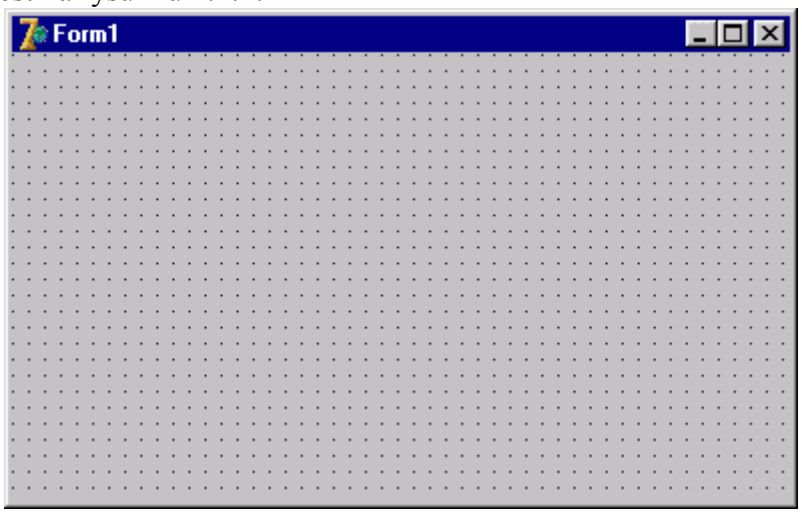
Paleta komponentów jest podzielona na zakładki, na których umieszczone są komponenty. Komponenty te są uporządkowane tematycznie. Pierwsza lista komponentów jaka jest widoczna po uruchomieniu Delphi, to komponenty umieszczone na zakładce **Standard** – rysunek 1.1.1.
Komponenty w Delphi są bardzo dużym ułatwieniem w czasie tworzenia aplikacji. Dzięki nim można znacząco ograniczyć pisanie kodu.



Rysunek 1.1.1. Widok palety komponentów

1.2. Formatki

Formatki są podstawowym elementem Delphi, na którym są umieszczane komponenty. W momencie uruchomienia Delphi formatka (ang. form) jest tworzona automatycznie stając się oknem głównym naszej aplikacji o nazwie domyślnej **Form1**. Wygląd formatki przedstawiony jest na rysunku 1.2.1.

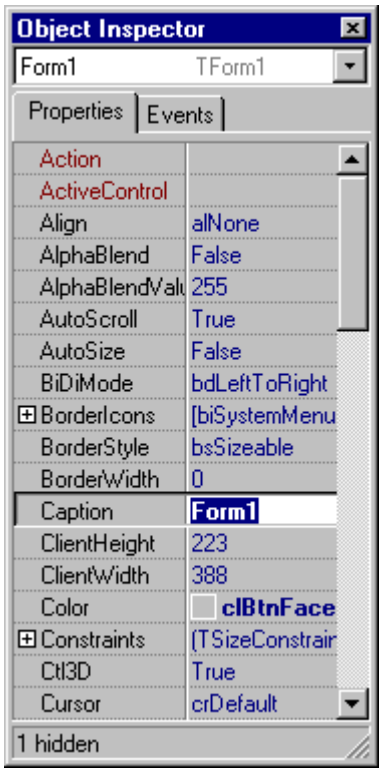


Rysunek 1.2.1. Widok formatki

W trakcie tworzenia programu komponenty są umieszczane na formatce. Widoczna siateczka na formatce umożliwia równomierne rozmieszczenie komponentów. Po uruchomieniu stworzonego programu siateczka jest nie widoczna.

1.3. Inspektor obiektów

Object Inspector (Inspektor obiektów) umożliwia zmianę właściwości komponentów umieszczonych na formatce lub samej formatki, w zależności od tego, co zostało wybrane. Właściwości formatki oraz wszystkich komponentów można zmieniać w kodzie programu. Wygląd Inspektora Obiektów ilustruje rysunek 1.3.1.



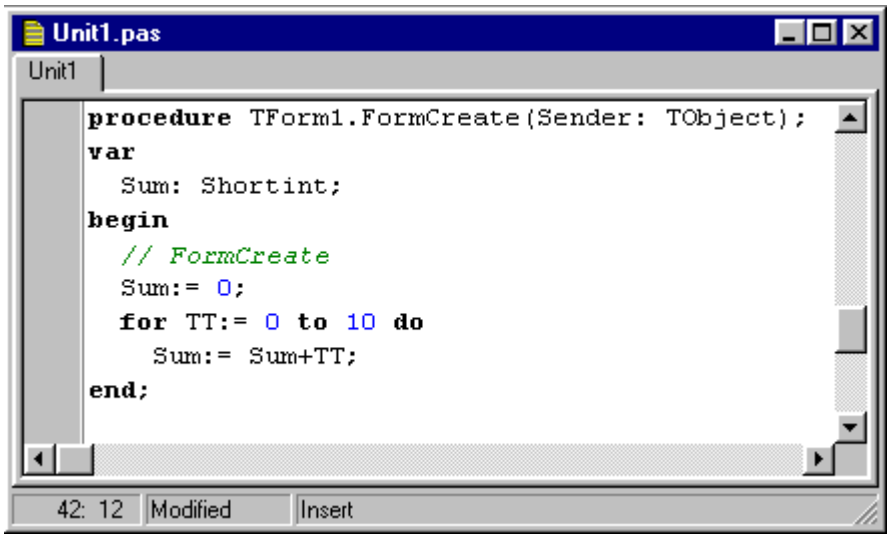
Rysunek 1.3.1. Widok okna Object Inspector (Inspektor obiektów) – zakładka Properties

Okno to podzielone jest na dwie zakładki. Zakładka pierwsza **Properties** (Właściwości) widoczna jest zaraz po uruchomieniu programu Delphi – rysunek 1.3.1 i zawiera właściwości aktualnie wybranego obiektu (formularza lub komponentu). Natomiast druga zakładka **Events** (Zdarzenia) zawiera listę zdarzeń przyporządkowaną danemu obiektowi (formularza lub komponentu). Dzięki tym zdarzeniom możliwe jest stworzenie obsługi np. kliknięcia klawisza. Każda zakładka jest podzielona na kolumny. Pierwsza kolumna posiada nazwy właściwości - zakładka **Properties** lub zdarzeń - zakładka **Events**. Druga kolumna każdej zakładki, umożliwia konfigurację wybranej właściwości lub wygenerowanie wybranego zdarzenia.

Powyżej zakładek znajduje się lista rozwijana, która zawiera listę nazw komponentów umieszczonych na formatce. Jakakolwiek zmiana nazwy komponentu jest automatycznie odzwierciedlana na tej liście. Klikając na klawisz (znajduje się z prawej strony listy rozwijanej) można rozwinąć listę.

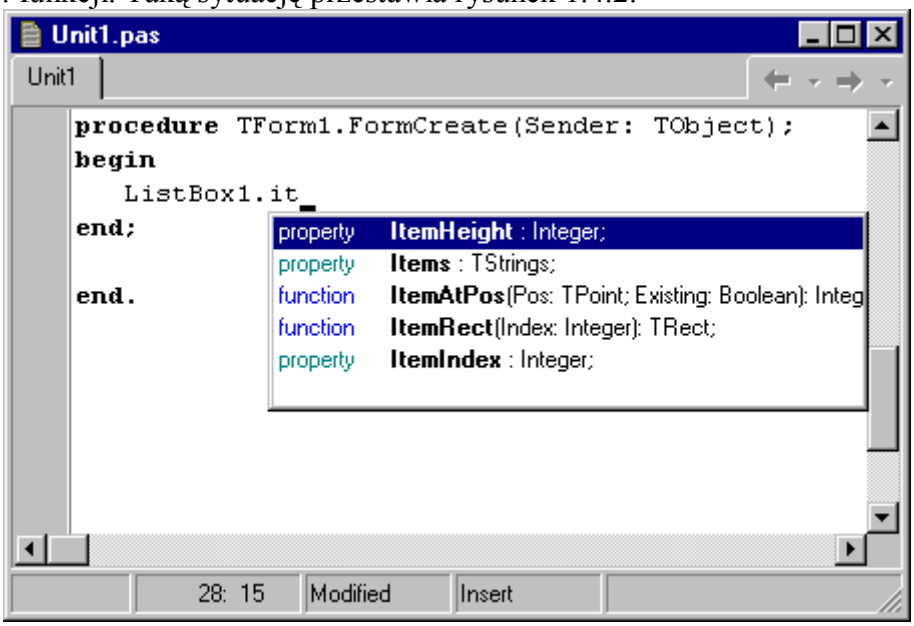
1.4. Edytor kodu

Edytor kodu jest edytorem tekstowym, w którym piszemy kod programu. Dla większej przejrzystości pisanego kodu, został on pokolorowany. Kolory te możemy zmieniać w opcjach programu. Wygląd tego edytora przedstawia rysunek 1.4.1.



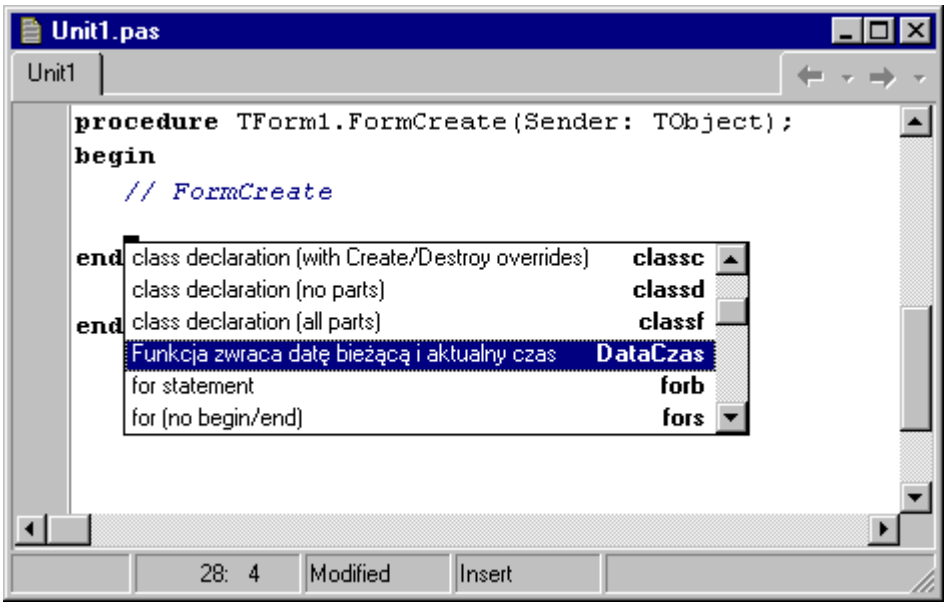
Rysunek 1.4.1. Widok edytora kodu

Edytor ten ma kilka udogodnień, o których warto wspomnieć. Pierwsze udogodnienie noszące nazwę **Code completion** (Uzupełnianie kodu) polega na uzupełnianiu wpisywanego kodu, przez co pisanie programu jest szybsze. Uzupełnianie kodu polega na wyświetleniu listy możliwych funkcji, procedur oraz właściwości komponentu lub modułu. Lista ta widoczna jest dopiero po wpisaniu pierwszej pełnej nazwy np. komponentu i po kropce pierwszych liter szukanej np. funkcji. Taką sytuację przestawia rysunek 1.4.2.



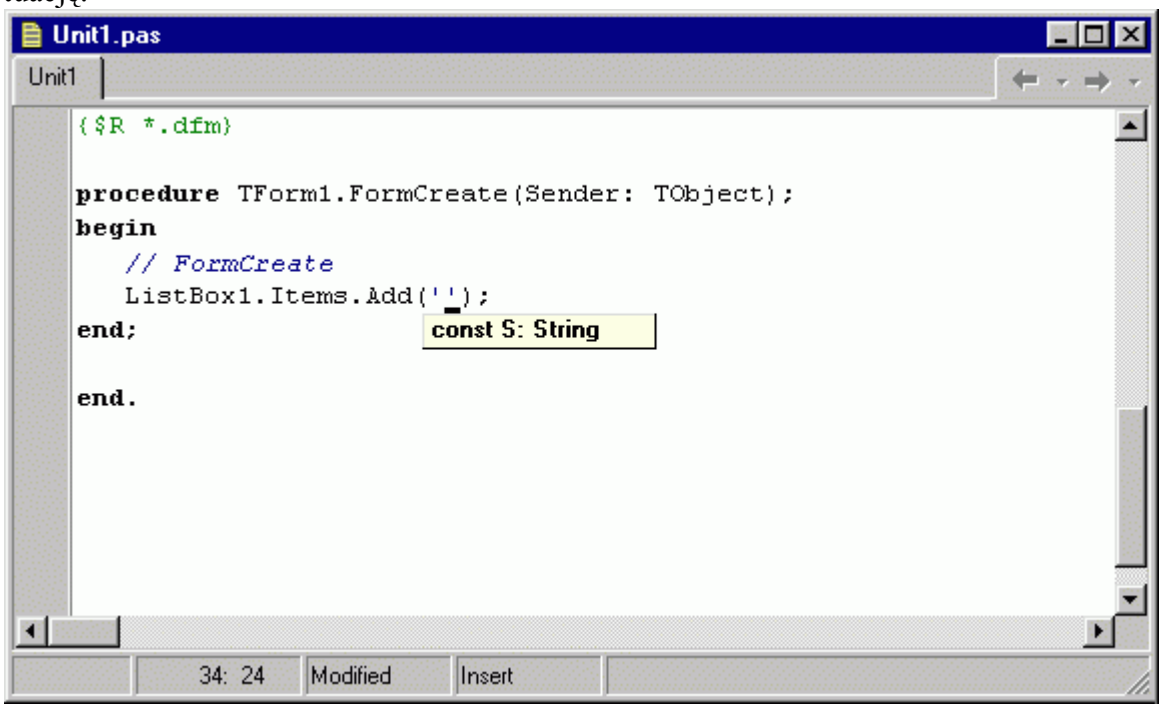
Rysunek 1.4.2. Widok działania funkcji uzupełniania kodu

Drugim udogodnieniem jest **Code templates** (Szablony kodów). Umożliwiają one wprowadzenie wybranego fragmentu kodu do pisanego programu, z listy wcześniej zdefiniowanych kodów. Listę zdefiniowanych kodów można wyświetlić poprzez wciśnięcie kombinacji klawiszy **CTRL+J**. Rysunek 1.4.3 przedstawia wyświetloną listę dostępnych szablonów kodu. Lista ta zapisywana jest w pliku tekstowym DELPHI32.DCI.



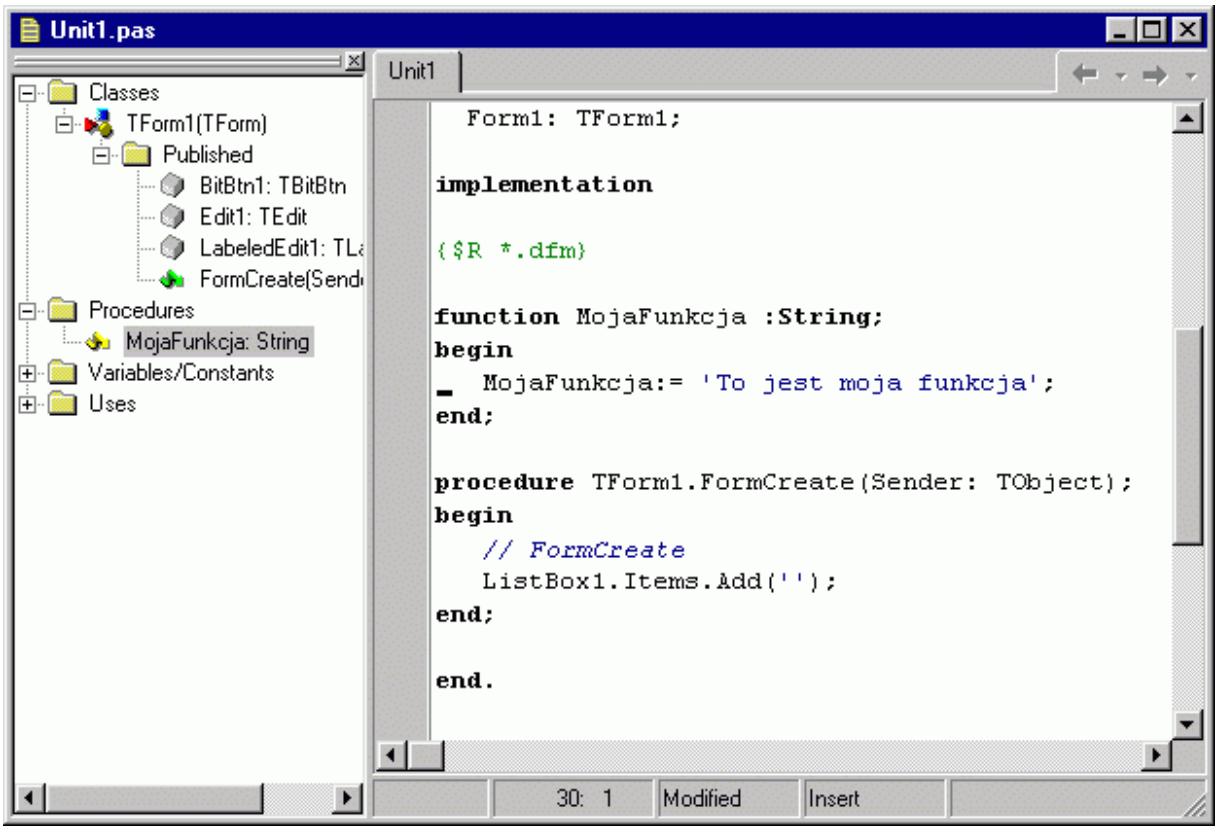
Rysunek 1.4.3. Widok listy szablonów kodu

Trzecim udogodnieniem wartym wspomnienia jest **Code parameters** (Parametry kodu). Dzięki temu udogodnieniu jesteśmy informowani o rodzaju parametru jaki należy umieścić we wcześniej napisanym rozkazie np. ListBox1.Items.Add(''); – zostanie wyświetlona informacja, że należy wpisać tekst – jeżeli kursor będzie ustawiony pomiędzy nawiasami. Informację tę również możemy uzyskać dzięki ustawieniu kursora w odpowiednim miejscu i naciśnięciu kombinacji klawiszy **CTRL+SHIFT+SPACJA**. Rysunek 1.4.4 przedstawia taką sytuację.



Rysunek 1.4.4. Widok informacji o parametrze funkcji

Następnym udogodnieniem jest **Code Explorer** (Eksplorator kodu). Służy on do nawigacji po zawartości pliku (modułu). Domyślną nazwą pliku jest Unit1. **Code Explorer** znajduje się (z reguły) po lewej stronie edytora zwierającego kod. Rysunek 1.4.5 przedstawia eksplorator kodu.

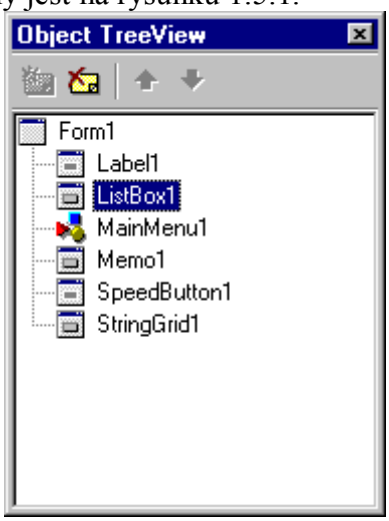


Rysunek 1.4.5. Widok Eksploratora kodu i edytora kodu

Eksplorator kodu umożliwia szybkie wyszukanie m.in. zmiennych, stałych, funkcji, procedur bez czasochłonnego przeszukiwania zawartości pliku (modułu).

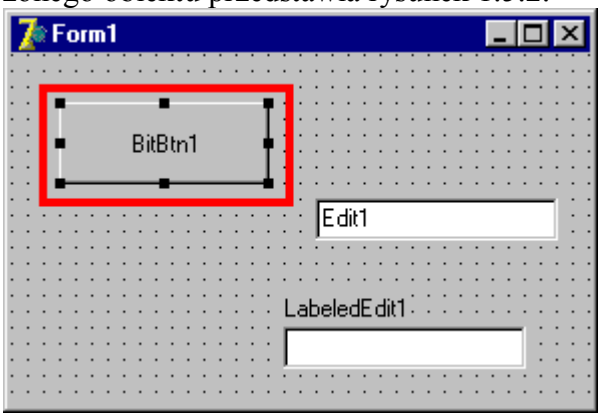
1.5. Drzewo obiektów

Object TreeView (Podgląd drzewa obiektów) służy do wyświetlania nazw obiektów umieszczonych na formatce. Za jego pomocą można przełączać się pomiędzy komponentami. Wygląd tego okna przedstawiony jest na rysunku 1.5.1.



Rysunek 1.5.1. Widok drzewa obiektów

Wyboru umieszczonego na formatce obiektu można dokonać również przez jednokrotne kliknięcie na dany obiekt. Aktywny (zaznaczony) obiekt jest wyróżniony tzw. uchwytami (jest ich osiem). Dzięki tym uchwytom można zmieniać szerokość i wysokość danego obiektu. Wygląd zaznaczonego obiektu przedstawia rysunek 1.5.2.



Rysunek 1.5.2. Widok zaznaczonego obiektu (tzn. klawisza)

2. Typy plików w Delphi

Delphi rozróżnia następujące typy plików:

- DPR – plik projektu (zawiera dane na temat modułów i formatek, które wchodzą w skład projektu);
- DFM – plik formatki (zawiera informacje o położeniu komponentów). Do każdego pliku jest tworzony plik modułu o tej samej nazwie;
- PAS – plik modułu (pliki te zawierają kody źródłowe wygenerowane przez Delphi oraz kody napisane przez programistę);
- DCU – plik skompilowany, który jest tworzony w momencie kompilacji;
- RES – plik zasobów (zawiera informacje na temat ikon, kursorów itp.);
- DSK – pliki te zawierają informację o wyglądzie środowiska i kompilacji;
- EXE – jest to plik skompilowany, który może być uruchomiony poza środowiskiem Delphi;
- DLL – są to biblioteki dołączane do programu dynamicznie w momencie pracy aplikacji;
- Pliki zapasowe o rozszerzeniu rozpoczynającym się znakiem tyldy (~) są tworzone w momencie zapisywania projektu na dysk.

3. Elementy Pascala

3.1. Komentarze

Komentarze umożliwiają opisanie fragmentów kodu, przez co staje się on czytelniejszy. W czasie kompilacji tekst komentarza jest pomijany. Delphi dysponuje następującymi rodzajami komentarzy:

- *{ komentarz }* – nawiasy klamrowe, które umożliwiają zaznaczenie znacznej części opisu znajdującego się w kilku liniach;

  przykład:
  {
  To jest komentarz.
  Taki sam był w Turbo Pascalu 7.0
  }

- *// komentarz* – umożliwia wstawienie komentarza tylko w jednej linii;

  przykład:
  // To jest komentarz, który można umieścić w jednej linii

- *(* to też jest komentarz *)* - nawiasy połączone z gwiazdką również umożliwiają zaznaczenie znacznej części opisu znajdującego się w kilku liniach;

  przykład:
  *(**
  To jest komentarz.
  Taki sam był w Turbo Pascalu 7.0
  **)*

3.2. Wybrane typy

Typ jest zbiorem określającym wartości, jakie może przyjmować zmienna lub stała.

Typ całkowity:

- Shortint – przyjmuje liczby z zakresu od –128 do 127 i ma rozmiar 1 bajta;
- Smallint – przyjmuje liczby z zakresu od –32768 do 32767 i ma rozmiar 2 bajtów;
- Longint – przyjmuje liczby z zakresu od –2147483648 do 2147483647 i ma rozmiar 4 bajtów;
- Byte – przyjmuje liczby z zakresu od 0 do 255 i ma rozmiar 1 bajta;
- Word – przyjmuje liczby z zakresu od 0 do 65535 i ma rozmiar 2 bajtów;
- Integer - przyjmuje liczby z zakresu od –2147483648 do 2147483647 i ma rozmiar 4 bajtów;
- Cardinal - przyjmuje liczby z zakresu od 0 do 4294967295 i ma rozmiar 4 bajtów;

Typy rzeczywiste:

- Real - przyjmuje liczby z zakresu od 2.9 x 10^-39 do 1.7 x 10^38 i ma rozmiar 6 bajtów (**uwaga**: typ ten został zachowany w celu zgodności z poprzednimi wersjami Pascala);
- Single - przyjmuje liczby z zakresu od 1.5 x 10^-45 do 3.4 x 10^38 i ma rozmiar 4 bajtów;
- Double - przyjmuje liczby z zakresu od 5.0 x 10^-324 od 1.7 x 10^308 i ma rozmiar 8 bajtów;
- Extended - przyjmuje liczby z zakresu od 3.4 x 10^-4951 do 1.1 x 10^4932 i ma rozmiar 10 bajtów;
- Comp - przyjmuje liczby z zakresu od –2^63+1 do 2^63 -1 i ma rozmiar 8 bajtów;
- Currency - przyjmuje liczby z zakresu od -922337203685477.5808 do 922337203685477.5807 i ma rozmiar 8 bajtów. Typ ten został stworzony na potrzeby obliczeń finansowych;
- Int64 – przyjmuje liczby z zakresu od -2^63 do 2^63-1.

Znak ^ oznacza potęgę.

Typ Variant:
Jest to dość niezwykłe rozwiązanie pozwalające na dynamiczną zmianę typu zmiennej, w trakcie działania programu (tzn. zmienna może przyjąć typ całkowity, rzeczywisty, itp.).

Przykład:

```
var
  V: Variant;
begin
  V:= 'To jest tekst'; // Zmienna „V" jest typem łańcuchowym
  V:= 1256; // Zmienna „V" jest typem całkowitym
end;
```

Typ logiczny:
Najczęściej używanym typem logicznym jest typ Boolean, który może przyjmować dwa stany. Pierwszy stan to TRUE (Prawda – logiczna 1), natomiast drugi to FALSE (Fałsz – logiczne 0).

3.3. Zmienne

Zmienne umożliwiają przechowywanie wartości liczbowych lub tekst, w zależności od zapotrzebowania. Zmienne znajdują się za słowem **var** (ang. Variables – zmienne), a przed blokiem instrukcji **begin** i **end** lub w sekcji **private** albo **public**.

Przykład:

```
var
  Hour, Min, Sec, MSec :Word; // Deklaracja zmiennych liczbowych

 // Deklaracja tablicy tekstowej składającej się z 4 elementów od 0 do 3
  txtTablica :array[0..3] of String;
```

Poniżej jest przedstawiony przykład prawidłowego umiejscowienia zmiennych w kodzie:

```
unit Unit1; // Nazwa modułu

interface // Część opisowa

uses // Umożliwia deklarację modułów (tzw. fragmentów kodu - biblioteka)
  Windows, Messages, SysUtils, Classes, Graphics, Forms;

type // Umożliwia deklarację różnego rodzaju komponentów, funkcji i procedur
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    // Deklaracja zmiennej „numLiczba” liczbowej typu całkowitego.
    // „numLiczba” jest zmienną globalną dla danego modułu np. Unit1
    numLiczba: Integer;

    // Deklaracja zmiennej tekstowej „txtTablica”, która jest tablicą
    // składającą się z 4 elementów i jest widoczna tylko w danym module np. Unit1
    txtTablica :array[0..3] of String;
  public
    { Public declarations }
    // Deklaracja zmiennej „txtString”, która ma przechować ciąg znaków
    // „txtString” jest zmienną globalną  dla całego projektu
    txtString: String;
  end;

var
  Form1: TForm1;

  // Zadeklarowanie zmiennej typu znakowego
  // Zmienna globalna widoczna w całym programie
  chrChar: Char;


implementation  // Umożliwia definicję funkcji i procedur

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
  okSwitch: Boolean;
// Deklaracja zmiennej logicznej „okSwitch”
// „okSwitch” jest zmienną lokalną widoczną tylko w
// funkcji, w której została zadeklarowana
begin
  // FormCreate

  // Przypisanie zmiennej „numLiczba” wartości 10
```

```
  numLiczba:= 10;

  // Przypisanie zmiennej lokalnej „okSwitch” wartość określającą prawdę
  okSwitch:= TRUE;

  // Przypisanie poszczególnym elementom tablicy ciągu znaków
   txtTablica[0]:= 'Mama';
   txtTablica[1]:= 'Jan';
   txtTablica[2]:= 'Adam';
   txtTablica[3]:= 'Tata';
end;

end.
```

Zmiennej tekstowej o nazwie „txtString” zadeklarowanej w module Unit1 w sekcji **public** możemy przypisać dowolny tekst z innego modułu (np. Unit2), ponieważ zmienna ta widoczna jest w całym programie. Oto przykład:

```
procedure TForm2.FormShow(Sender: TObject);
begin
  // FormShow
  Form1.txtString:= 'Tester';
end;
```

3.4. Stałe

Stałe wprowadza się do programu w celu ułatwienia późniejszej modyfikacji programu, którą możemy wykonać w jednym miejscu. W przeciwnym przypadku musielibyśmy wprowadzać zmiany w kilku miejscach, co może przyczynić się do powstawania błędów. Stałą deklarujemy po słowie **const** (ang. Constants – stałe). Oto przykłady:

```
const
   Miesiące = 12; // Deklaracja stałej „Miesiące” określającej ilość miesięcy w roku
   Doba = 24; // Deklaracja stałej „Doba” określającej ilość godzin w ciągu doby

   // Deklaracja stałej tekstowej „txtDniTygodnia”, która jest tablicą
   // składającą się z 7-miu elementów
   txtDniTygodnia :array[0..6] of String = ('Poniedziałek',
                          'Wtorek', 'Środa', 'Czwartek',
                          'Piątek', 'Sobota', 'Niedziela');
```

Poniżej jest przedstawiony przykład umiejscowienia stałych w kodzie:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Forms;
```

```
const
  // Deklaracja stałej liczbowej „numLiczba”, której została przypisana wartość 23 i
 // jest widoczna w danym module
  numLiczba = 23;
type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;


implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
const
  // Deklaracja stałej tekstowej „txtString”, która widoczna jest tylko w funkcji,
 // w której została zadeklarowana
  txtString = 'Jan Biernat';
begin
  // FormCreate
end;

end.
```

3.5. Instrukcje warunkowe IF...Then / IF...Then...Else

Instrukcje warunkowe **if**...**then** / **if**...**then**...**else** pozwalają na sterowanie programem. Sterowanie to polega na spełnieniu określonego warunku, po spełnieniu którego zostanie wykonana odpowiednia część programu (kodu). Jeżeli chce się wywołać większą ilość funkcji, procedur lub wpisać większą ilość instrukcji to trzeba je wpisać pomiędzy słowami **begin** i **end**. W przypadku wywołania jednej funkcji, procedury lub wpisania jednej instrukcji, to słowa **begin** i **end** można pominąć.

Oto konstrukcja instrukcji warunkowych:

```
if (warunek) then // Wykonanie funkcji w momencie spełnienia warunku
   funkcja;
```

lub

```
if (warunek) then // Wykonanie funkcji i procedury w momencie spełnienia warunku
begin
   funkcja;
   procedura;
   instrukcje;
end;
```

lub

```
if (warunek) then // Wykonanie funkcji w momencie spełnienia warunku
   funkcja
else      // Wykonanie funkcji w momencie spełnienia drugiego warunku
   funkcja2;
```

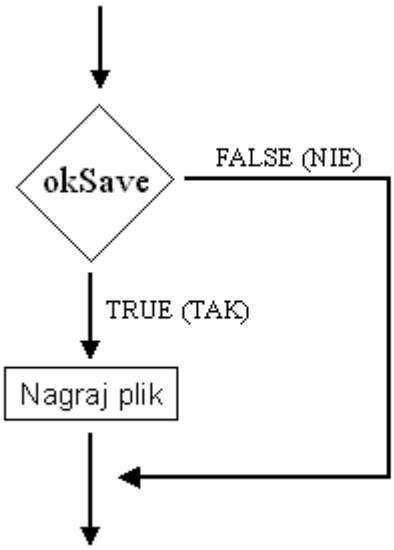
lub

```
if (warunek) then
begin
   funkcja;
   procedura;
   instrukcje;
end
else
begin
   funkcja;
   procedura;
   instrukcje;
end;
```

Przykład 1:

```
  if (okSave = TRUE) then ZapiszPlik('nazwa');
```

Schemat blokowy dla przykładu 1 ilustrujący instrukcję warunkową **IF...THEN**:

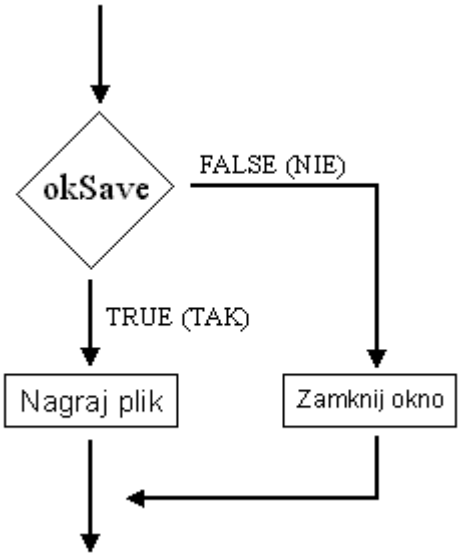


Jeżeli zmienna logiczna „okSave” będzie prawdziwa (czyli będzie miała wartość TRUE), to spełniony będzie warunek, a co za tym idzie nastąpi nagranie pliku.

Przykład 2:

if (okSave = TRUE) **then**
savZapisz(‘nazwa’) *// Następuje zapisanie w momencie spełnienia warunku*
else
Close; *// Następuje zamknięcie okna w przypadku spełnienia drugiego warunku*

Schemat blokowy dla przykładu 2 ilustrujący instrukcję warunkową **IF...THEN...ELSE**:



Jeżeli zmienna logiczna „okSave” będzie prawdziwa (czyli będzie miała wartość TRUE), to spełniony będzie warunek, a co za tym idzie nastąpi nagranie pliku. W przeciwnym przypadku zmienna logiczna „okSave” będzie nieprawdziwa (czyli będzie miała wartość

FALSE). Wtedy to zostanie spełniony warunek drugi i nastąpi wykonanie instrukcji, które są zawarte po słowie **else** (w tym przykładzie nastąpi zamknięcie okna).

3.6. Instrukcja wiążąca With...Do

Instrukcja wiążąca jest m.in. przydatna w momencie wywoływania kilku właściwości komponentu. Ta konstrukcja zwalnia programistę z ciągłego wypisywania nazwy komponentu w trakcie wywołania właściwości danego komponentu.

Przykład bez użycia instrukcji wiążącej **with...do**:

```
Memo1.ReadOnly:= TRUE;
Memo1.TabStop:= FALSE;
Memo1.ScrollBars:= ssNone;
Memo1.Color:= clBtnFace;
Memo1.Lines.Clear;
```

Przykład z użyciem instrukcji wiążącej **with...do**:

```
with Memo1 do
begin
  ReadOnly:= TRUE;
  TabStop:= FALSE;
  ScrollBars:= ssNone;
  Color:= clBtnFace;
  Lines.Clear;L
  Lines.Add('');
end;
```

Z powyższych przykładów widać, że bardziej efektywnym rozwiązaniem jest użycie konstrukcji wiążącej **with...do**.

3.7. Pętle For...To/Downto... Do, While...Do, Repeat…Until

Instrukcje **for..to**/**downto...do**, **while...do**, **repeat...until** umożliwiają wykonanie instrukcji w sposób cykliczny, tzn. z góry określoną ilość razy.
Do zatrzymania wykonywanej pętli służy instrukcja **break**, która powoduje zakończenie wykonywanej pętli, w której została wywołana.

- **for**...**to**...**do** – wykonuje blok instrukcji określoną ilość razy np. zwiększając zmienną TT z wartości 0 do wartości 10.

```
for TT:= 0 to 10 do
   Sum:= Sum+1;
```

lub

```
for TT:= 0 to 10 do
```

```
begin
   Instrukcja 1;
   Instrukcja 2;
   ……————;
   Instrukcja N;
end;
```

- **for**…**downto**…**do** – wykonuje blok instrukcji określoną ilość razy np. zmniejszając zmienną TT z wartości 10 do wartości 0.

```
for TT:= 10 downto 0 do
   Sum:= Sum+1;
```

lub

```
for TT:= 10 downto 0 do
begin
   Instrukcja 1;
   Instrukcja 2;
   ……————;
   Instrukcja N;
end;
```

- **while**...**do** – wykonuje blok instrukcji tak długo, jak długo spełniony jest warunek. Warunek ten jest sprawdzany na początku pętli przy każdym cyklu. W przypadku nie spełnienia warunku, wykonywanie pętli jest zatrzymane. Zdarzyć się może, że pętla nie zostanie wykonana ani razu, ponieważ warunek, który jest sprawdzany na początku przed wykonaniem pętli nie został spełniony.

```
while (warunek) do
    instrukcja;
```

lub

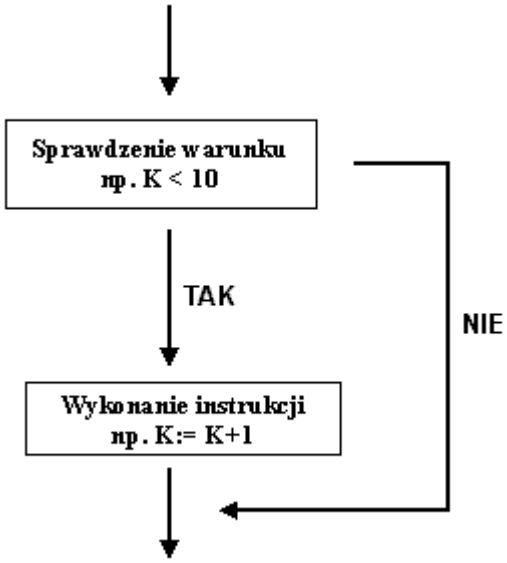
```
while (warunek) do
begin
   Instrukcja 1;
   Instrukcja 2;
   ……————;
   Instrukcja N;
end;
```

Przykład:

```
while (K < 10) do
begin
   K:= K+1;
```

end;

Schemat blokowy dla powyższego przykładu



Jeżeli zmienna „K” jest mniejsza od liczby 10, to spełniony będzie warunek i nastąpi zwiększenie zmiennej „K” o wartość 1. Zmienna „K” będzie tak długo zwiększana, dopóki będzie spełniony warunek „K<10”. W przypadku niespełnienia warunku wykonywanie pętli zostanie zatrzymane.

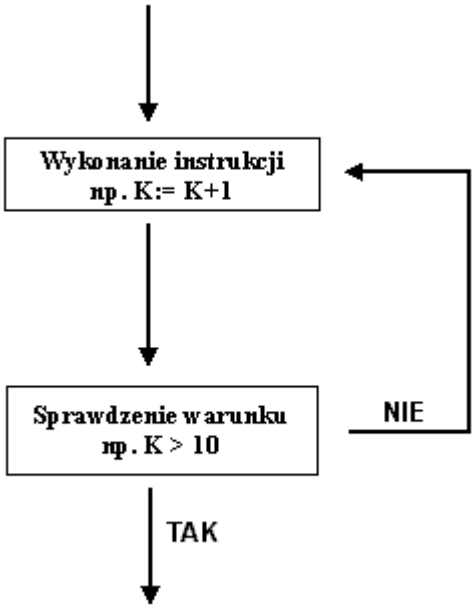
- **repeat**...**until** – wykonuje blok instrukcji tak długo, aż zostanie spełniony warunek. Warunek ten jest sprawdzany na końcu każdego cyklu. Sprawdzenie warunku na końcu cyklu powoduje wykonanie pętli przynajmniej jeden raz.

```
repeat
   Instrukcja 1;
   Instrukcja 2;
   ……————;
   Instrukcja N;
until(warunek);
```

Przykład:

```
repeat
   K:= K+1;
until(K > 10);
```

Schemat blokowy dla powyższego przykładu:



Zwiększaj zmienną „K” o wartość 1, tak długo, aż zmienna „K” będzie większa od liczby 10. Gdy warunek będzie spełniony, to przerwij wykonywanie pętli.

Przykład znajduje się na dyskietce w katalogu D7CW\PETLE.

3.8. Mechanizmy obsługi wyjątku

- **try...except...end** - mechanizm obsługi wyjątku jest bardzo wygodnym narzędziem pozwalającym na wychwycenie sytuacji wyjątkowych (np. dzielenie przez zero). Dzięki temu mechanizmowi program jest bardziej stabilny. Mechanizm wyjątku działa tylko w przypadkach wystąpienia błędu.
  UWAGA: Program z mechanizmem obsługi błędów należy uruchomić poza środowiskiem Delphi, w przeciwnym razie obsługa błędu będzie przechwycona przez Delphi.

  Konstrukcja:

```
try
    ...
    instrukcje mogące spowodować błąd
    ...
except
    ...
    instrukcje wykonywane po wystąpieniu błędu
    ...
end;
```

Przykład znajduje się na dyskietce w katalogu D7CW\TRY_EX.

- **try...finally...end** - mechanizm zwalniania zasobów gwarantuje zwolnienie zasobów (np. plik, pamięć dynamiczna, zasoby systemowe i obiekty) w przypadku wystąpienia błędu. Instrukcje zawarte w bloku zwalniania zasobów są wykonywane zawsze. Blok zwalniania zasobów następuje po słowie **finally**.

  Konstrukcja:

```
try
  ...
  instrukcje korzystające z zasobów i mogące spowodować błąd
  ...
finally
  ...
  zwalnianie zasobów (instrukcje tu zawarte wykonywane są zawsze).
  ...
end;
```

Przykład znajduje się na dyskietce w katalogu D7CW\TRY_FI.

3.9. Opis wybranych zdarzeń

Niżej opisane zdarzenia dotyczą formatki, aczkolwiek niektóre z nich występują w różnych komponentach. Zdarzenia te powodują wykonanie pewnych instrukcji po zajściu określonego zdarzenia, które jest wykonane w tym przypadku na formatce. Nazwa zdarzeń na liście właściwości **Object Inspector** (Inspektora obiektów) jest inna niż wewnątrz kodu, np. **OnCreate** na liście właściwości Inspektora Obiektu jest równoznaczna z nazwą **FormCreate** wewnątrz kodu – odnosi się to do nazw wszystkich zdarzeń.

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  // Tu wpisujemy instrukcje, które są wykonywane w momencie gdy formatka jest aktywna.
  // np. instrukcje sumowania dwóch liczb, wyświetlania komunikatu, itp.
end;

procedure TForm1.FormClick(Sender: TObject);
begin
  // Tu wpisujemy instrukcje, które są wykonywane w momencie gdy kliknie się na formatce.
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  // Tu wpisujemy instrukcje, które są wykonywane w momencie zamykania formatki.
end;

procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  {
   Tu wpisujemy instrukcje, które mają na celu zadanie pytania użytkownikowi w
   momencie zamykania formatki np. wywołanie dialogu z pytaniem.
```

```
  }
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // Tu wpisujemy instrukcje, które są wykonywane przed utworzeniem okna formatki.
end;

procedure TForm1.FormDblClick(Sender: TObject);
begin
  // Tu wpisujemy instrukcje, które są wykonywane w momencie
  // dwukrotnego kliknięcia na formatce.
end;

procedure TForm1.FormDeactivate(Sender: TObject);
begin
  // Tu wpisujemy instrukcje, które są wykonywane w momencie,
  // gdy formatka przestaje być aktywna.
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  // Tu wpisujemy instrukcje, które są wykonywane w momencie likwidacji formatki.
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  // Tu wpisujemy instrukcje, które są wykonywane w momencie
  // naciśnięcia klawiszy funkcyjnych np. Enter, F1..F12, PageUp itp.
end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  // Tu wpisujemy instrukcje, które są wykonywane w momencie naciśnięcia
  // dowolnego lub wybranego klawisza alfanumerycznego np. a, b, <, > itp.
end;

procedure TForm1.FormResize(Sender: TObject);
begin
  // Tu wpisujemy instrukcje, które są wykonywane w momencie
  // zwiększania lub zmniejszania rozmiaru formatki.
end;

procedure TForm1.FormShow(Sender: TObject);
begin
  // Tu wpisujemy instrukcje, które są wykonywane w momencie ukazania się formatki.
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
```

```
  Y: Integer);
begin
  // Tu wpisujemy instrukcje, które wykonywane są w momencie poruszania kursorem myszy.
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  {
   Tu wpisujemy instrukcje odpowiedzialne za odświeżenie zawartości
  formatki w przypadku zasłonięcia tej formatki przez inną formatkę.
  }
end;
```

Przedstawiony przykład znajduje się w katalogu D7CW\OPIS.

3.10. Lista wybranych zmiennych globalnych

Poniżej znajduje się opis wybranych zmiennych globalnych, które są dostępne w całym programie stworzonym w środowisku Delphi.

Do tych zmiennych przypisywane są wartości, pobrane z systemu w momencie uruchomienia programu. Wartości te można zmieniać, bez wpływu na zmiany ustawień w systemie. Zmienne te ułatwiają bardzo prace nad programem, ponieważ dają możliwość określenia pewnych ustawień dla tworzonego programu. Dzięki temu, program działa według określonych ustawień przez twórcę, nie dokonując zmian w ustawieniach systemu.

- **DateSeparator** – umożliwia przypisanie lub odczyt znaku używanego w zapisie daty do oddzielenia roku, miesiąca, dnia (np. DateSeparator:= '-' – od tego momentu znakiem rozdzielającym datę jest znak minus /2003-04-14/);
- **TimeSeparator** - umożliwia przypisanie lub odczyt znaku używanego w zapisie czasu do oddzielenia godzin, minut, sekund (np. TimeSeparator:= ':' – od tego momentu znakiem rozdzielającym czas jest znak dwukropek /23:45:12/);
- **DecimalSeparator** - umożliwia przypisanie lub odczyt znaku używanego w zapisie liczbowym do oddzielenia liczby całkowitej i jej części dziesiętnej (np. DecimalSeparator:= '.' – od tego momentu znakiem rozdzielającym liczbę całkowitą od jej dziesiętnej części jest kropka).
  UWAGA: W językach programowania m.in. w Delphi używa się kropki (np. 12.34 – poprawny zapis) zamiast przecinka, w celu oddzielenia liczby całkowitej od jej części dziesiętnej. Gdy używa się przecinek, program ze względu na błąd nie uruchomi się (np. 12,34 – jest zapisem błędnym);
- **ShortMonthNames** – Tablica znaków zawierająca skrócone nazwy miesiąca (np. Caption:= ShortMonthNames[1]; - zwróci nam skróconą nazwę pierwszego miesiąca „sty");
- **LongMonthNames** – Tablica znaków zawierająca pełne nazwy miesiąca (np. Caption:= LongMonthNames[2]; - zwróci nam pełną nazwę drugiego miesiąca „luty");
- **ShortDayNames** – Tablica znaków zawierająca skrócone nazwy dnia (np. Caption:= ShortDayNames[1]; - zwróci nam skróconą nazwę pierwszego dnia „po");
- **LongDayNames** – Tablica znaków zawierająca pełne nazwy dnia (np. Caption:= LongDayNames[2]; - zwróci nam pełną nazwę drugiego dnia „wtorek");

Przykład zastosowania wyżej wymienionych zmiennych:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // FormCreate
  // Określenie znaku, który będzie służył do oddzielenia roku, miesiąca, dnia.
  DateSeparator:= '-';


  // Określenie znaku, który będzie służył do oddzielenia liczby całkowitej
  // od jej części dziesiętnej.
  DecimalSeparator:= '.';

  // Zwraca skróconą nazwę pierwszego miesiąca.
  Label1.Caption:= ShortMonthNames[1];

  // Zwraca pełną nazwę drugiego miesiąca.
  Label2.Caption:= LongMonthNames[2];

  // Zwraca skróconą nazwę pierwszego dnia.
  Label3.Caption:= ShortDayNames[1];

  // Zwraca pełną nazwę drugiego dnia.
  Label4.Caption:= LongDayNames[2];
end;
```

3.11 Operatory logiczne

Operatory logiczne są wykorzystywane do testowania (sprawdzania) jednocześnie kilku warunków. Poniżej przedstawione są operatory wraz z przykładami ilustrującymi ich zastosowanie.

Operator	Wygląd
Przypisania	:=
Równości	=
Nierówności	<>
Mniejszości	<
Większości	>
Większe lub równe	>=
Mniejsze lub równe	<=
Logiczne i	and
Logiczne lub	or
Zaprzeczenie	not

Przykład przypisania:

```
Liczba := 5
```

Przypisanie zmiennej o nazwie „Liczba” wartości 5.

Przykład równości:

if (Numer = 5) **then** WykonajZadanie

Jeżeli zmienna o nazwie „Numer” jest równa liczbie 5 to wykonaj instrukcję po słowie **THEN**.

Przykład nierówności:

if (Numer <> 5) **then** WykonajZadanie

Jeżeli zmienna o nazwie „Numer” jest różna od liczby 5 to wykonaj instrukcję po słowie **THEN**.

Przykład większe lub równe:

if (Numer >= 5) **then** WykonajZadanie

Jeżeli zmienna o nazwie „Numer” jest większa od liczby 5 lub jest równa liczbie 5 to wykonaj instrukcję po słowie **THEN**.

Przykład zastosowania operatora AND:

if (warunek1) **and** (warunek2) **then** WykonajZadanie

Jeżeli będzie spełniony Warunek1 i Warunek2 to wykonaj instrukcję po słowie **THEN**.

Przykład zastosowania operatora OR:

if (warunek1) **or** (warunek2) **then** WykonajZadanie

Jeżeli będzie spełniony Warunek1 lub Warunek2 to wykonaj instrukcję po słowie **THEN**.

Przykład zastosowania operatora NOT:

```
while not Warunek do
begin
  WykonajZadanie;
end;
```

Instrukcja pomiędzy słowem **Begin** i **End** jest wykonana w momencie spełnienia warunku postawionego na samym początku pętli.

3.12 Obsługa klawiatury

Z obsługi zdarzeń związanych z klawiaturą korzystamy, gdy chcemy w programie przechwycić naciśnięcie jakiegoś klawisza. Zdarzenia związane z klawiaturą posiadają niektóre komponenty (np. Memo1) oraz formatka. Do obsługi klawiatury alfanumerycznej służy m.in. zdarzenie **OnKeyPress** (w kodzie programu to zdarzenie nosi nazwę **FormKeyPress**).

W celu obsługi np. klawisza ESC (znajduje się w lewym górny rogu klawiatury) i ENTER na leży w zdarzeniu **OnKeyPress** napisać następujące linie.

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  // Przykład obsługi klawisza ESC
  if (Key = CHR(27)) then Close;

  // Przykład obsługi klawisza ENTER
  if (Key = CHR(13)) then ShowMessage('Został naciśnięty klawisz ENTER.');
end;
```

Do obsługi klawiszy służą jeszcze dwa inne zdarzenia **OnKeyDown** i **OnKeyUp**. Zdarzenia **OnKeyDown** występuje, gdy został naciśnięty dowolny klawisz. Natomiast zdarzenie **OnKeyUp** występuje, gdy zwolniony został dowolny klawisz.

W celu obsługi klawiszy strzałkowych, kombinacji **CTRL+ALT+SHIFT+Q** oraz **CTRL+Q** należy w zdarzeniu **OnKeyDown** wpisać następujące linie:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  // Przykład obsługi kombinacji klawiszy CTRL+ALT+SHIFT+Q
  if (Shift = [ssCtrl, ssAlt, ssShift]) then
  begin
    if (CHR(Key) in ['Q', 'q']) then
    begin
      ShowMessage('Nacisnąłeś kombinację klawiszy CTRL+ALT+SHIFT+Q.');
    end;
  end
  else
  if ((ssCtrl in Shift) and (CHR(Key) in ['Q', 'q'])) then
  begin
    // Przykład obsługi kombinacji CTRL+Q
    ShowMessage('To jest kombinacja CTRL+Q');
  end;
end;
```

Poniżej zamieszczona jest tabela z wybranymi wartościami parametru **Shift**:

Wartość	Znaczenie
ssAlt	Klawisz ALT
ssShift	Klawisz SHIFT
ssCtrl	Klawisz CTRL

Poniżej zamieszczona jest tabela z wybranymi wartościami parametru **Key**:

Kod	Wartość	Znaczenie
VK_CANCEL	3	Kombinacja CTRL+BREAK
VK_BACK	8	Klawisz BACKSPACE
VK_TAB	9	Klawisz TAB
VK_RETURN	13	Klawisz ENTER

VK_CLEAR	12	Klawisz NUMLOCK
VK_SHIFT	16	Klawisz SHIFT
VK_CONTROL	17	Klawisz CTRL
VK_MENU	18	Klawisz ALT
VK_ESCAPE	27	Klawisz ESC
VK_SPACE	32	Klawisz SPACE (spacja)
VK_PRIOR	33	Klawisz PAGEUP
VK_NEXT	34	Klawisz PAGEDOWN
VK_END	35	Klawisz END
VK_HOME	36	Klawisz HOME
VK_LEFT	37	Klawisz: strzałka w lewo
VK_UP	38	Klawisz: strzałka w górę
VK_RIGHT	39	Klawisz: strzałka w prawo
VK_DOWN	40	Klawisz: strzałka w dół
VK_INSERT	45	Klawisz INSERT
VK_DELETE	46	Klawisz DELETE
VK_0 ... VK_9	48...57	Klawisze cyfr od 0 do 9-ciu
VK_A ... VK_Z	65...90	Klawisze liter od A do Z
VK_LWIN	91	Klawisz: lewy WIN
VK_RWIN	92	Klawisz: prawy WIN
VK_NUMPAD0 ... VK_NUMPAD9	96...105	Blok numeryczny od 0 do 9
VK_F1 ... VK_F12	112...123	Klawisze funkcyjne F1 ... F12
VK_SCROLL	145	Klawisz SCROLL LOCK
VK_LSHIFT	160	Klawisz: lewy SHIFT
VK_RSHIFT	161	Klawisz: prawy SHIFT
VK_LCONTROL	162	Klawisz: lewy CTRL
VK_RCONTROL	163	Klawisz: prawy CTRL
VK_CAPITAL	20	Klawisz CAPS LOCK
VK_SNAPSHOT	44	Klawisz PRINT SCREEN

W celu usunięcia informacji o naciśnięciu jakiegoś klawisza np. ENTER wystarczy w zdarzeniu (obsługującym naciskanie klawiszy) ustawić parametr **Key** na wartość CHR(0). Jest to przydatne w przypadku, gdy na formatce znajduje się tzw. klawisz domyślny (to taki klawisz, który można użyć za pomocą klawisza ENTER będąc na dowolnym elemencie formatki), a mimo tego zachodzi potrzeba obsłużenia klawisza ENTER w jakimś komponencie. Dzięki takiemu rozwiązaniu klawisz domyślny nie zostanie uaktywniony. Poniższy przykład ilustruje wyzerowanie parametru **Key**:

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  // Przykład obsługi klawisza ENTER
  if (Key = CHR(13)) then
  begin
     // W momencie naciśnięcia klawisza ENTER wyzeruj parametr KEY
     Key:= CHR(0);
  end;
end;
```

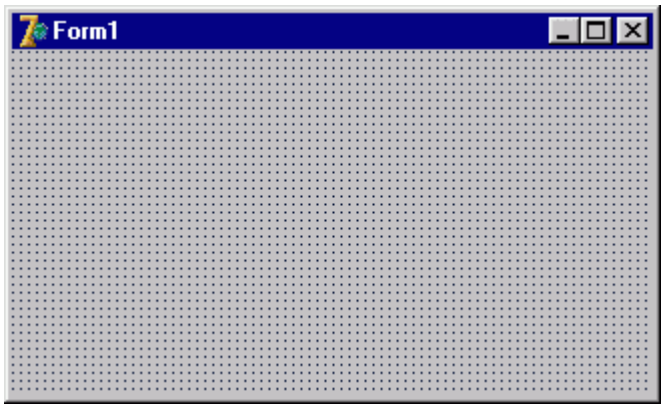
4. Ćwiczenia ze środowiska programistycznego DELPHI 7

Poniżej zamieszczone są wybrane ćwiczenia dotyczące środowiska programistycznego Borland DELPHI 7.0, które mają pomóc w oswojeniu się z programem DELPHI.

Ćwiczenie 4.1. Zwiększenie ilości punktów siatki

Zwiększ ilość punktów siatki.

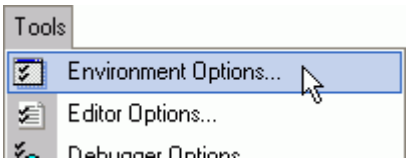
Opis:
Siatka widoczna w postaci punktów umożliwia łatwiejsze układanie komponentów na formie. Siatka ta jest widoczna tylko w trakcie tworzenia programu. Wygląd siatki przedstawia rysunek 4.1.1.



Rysunek 4.1.1. Widok siatki na formie

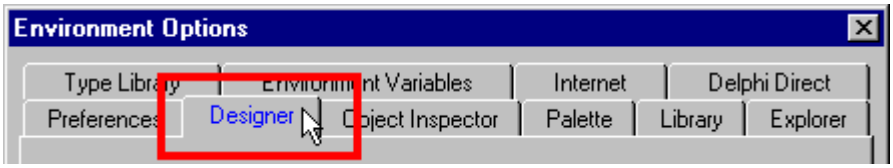
Sposób wykonania:

- Wybierz menu **Tools/Environment Options...** (Narzędzia/Opcje Środowiskowe) – rysunek 4.1.2;



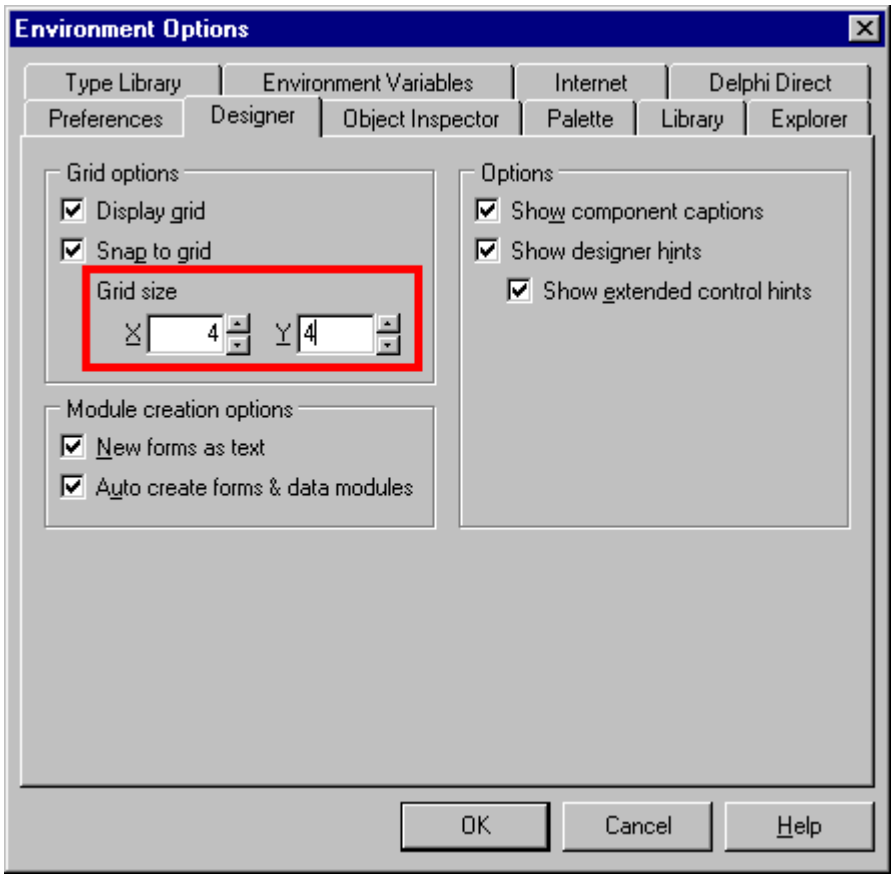
Rysunek 4.1.2. Widok wybranego polecenia „Environment Options...”

- W oknie **Environment Options**(Opcje Środowiskowe) wybierz zakładkę **Designer** (Projektant) – rysunek 4.1.3;



Rysunek 4.1.3. Widok wybranej zakładki „Designer (Projektant)”

- W sekcji **Grid Options**(Opcje siatki) w polach **Grid Size X** i **Y** (Rozmiar siatki) wpisz wartość 4 dla obydwóch pól – rysunek 4.1.4;



Rysunek 4.1.4. Widok edytowania pól „Grid Size XY(Rozmiar siatki XY)”

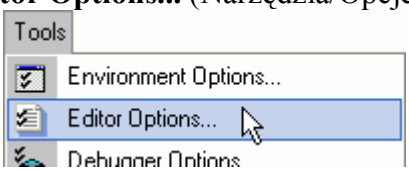
- Po wpisaniu odpowiednich wartości kliknij na klawisz **OK**, w celu zatwierdzenia ustawień.

Ćwiczenie 4.2. Wybór koloru edytora oraz składni

Zmień kolor edytora oraz składni, tak przypominał ustawienia kolorów z Turbo Pascal 7.0.

Sposób wykonania:

- Wybierz menu **Tools/Editor Options...** (Narzędzia/Opcje edytora) – rysunek 4.2.1;

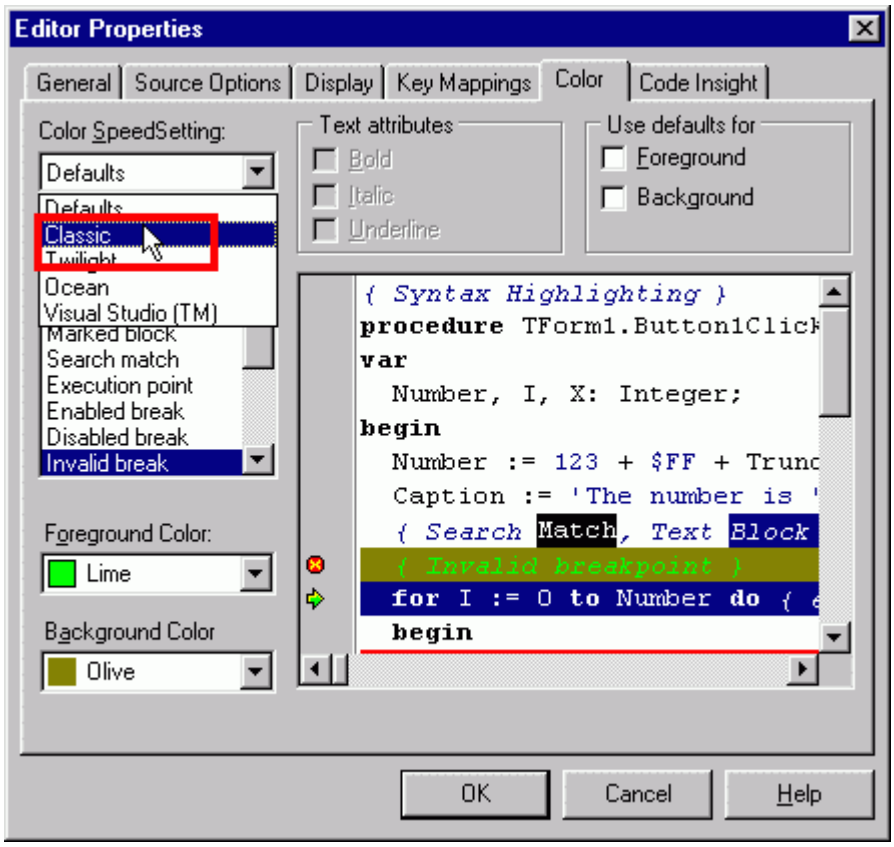


Rysunek 4.2.1. Widok wybranego polecenia Editor Options

- W oknie **Editor Properties** (Właściwości edytora) wybierz zakładkę **Color** (Kolor) – 4.2.2;

Rysunek 4.2.2. Widok wybranej zakładki Color (Kolor)

- Rozwiń listę **Color Speed Setting** (Szybkie ustawienia koloru) – rysunek 4.2.3;
- Wybierz z tej listy pozycję **Classic** (Klasyczny) – rysunek 4.2.3;



Rysunek 4.2.3. Widok rozwiniętej listy Color Speed Setting (Szybkie ustawienia koloru) oraz wybranej pozycji Classic (Klasyczny)

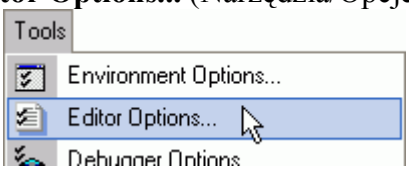
- Po wybraniu schematu koloru kliknij na klawisz OK, aby zatwierdzić ustawienia.

Ćwiczenie 4.3. Szablon kodu

Wprowadź do listy szablonów kodu fragment kodu umożliwiającego wyświetlenie bieżącej daty i aktualnego czasu.

Sposób wykonania:

- Wybierz menu **Tools/Editor Options...** (Narzędzia/Opcje edytora) – rysunek 4.3.1;



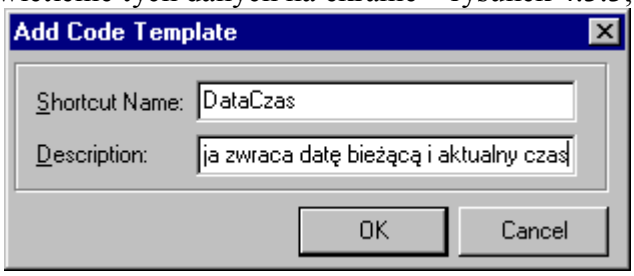
Rysunek 4.3.1. Widok wybranego polecenia Editor Options

- W oknie **Editor Properties** (Właściwości edytora) wybierz zakładkę **Source Options** (Opcje kodu) – rysunek 4.3.2;

Rysunek 4.3.2. Widok wybranej zakładki „Source Options (Opcje kodu)”

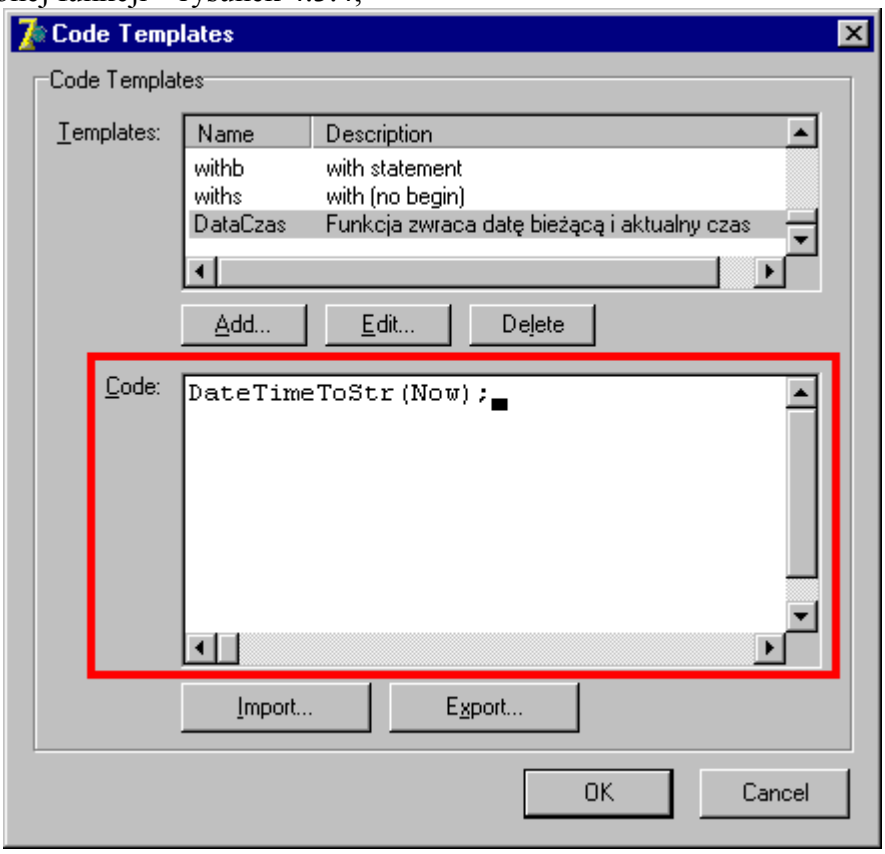
- Kliknij na klawisz **Edit Code Templates** (Edytuj szablony kodów);
- W oknie **Code Templates** (Szablony kodów) kliknij na klawisz **Add**(Dodaj);
- W oknie **Add Code Template** (Dodaj szablon kodu) w polu **Shortcut Name** (Skrócona nazwa) wpisz krótką nazwę funkcji, np. „DataCzas”. Natomiast w polu **Description** (Opis) wpisz krótki opis mówiący o przeznaczeniu funkcji, np. „Funkcja

zwraca datę bieżącą i aktualny czas", tzn. następuje pobranie aktualnej daty i czasu z komputera i wyświetlenie tych danych na ekranie – rysunek 4.3.3;



Rysunek 4.3.3. Widok okna „Add Code Template" z wypełnionym polami

- Po wypełnieniu pól kliknij na klawisz **OK**, co spowoduje dodanie nowej funkcji do listy **Templates** (Szablony) i przeniesie kursor do pola edycyjnego **Code** (Kod);
- Będąc w polu edycyjnym **Code** (Kod) wpisz kod np. **DateTimeToStr(Now)** dla nowo stworzonej funkcji – rysunek 4.3.4;



Rysunek 4.3.4. Widok wprowadzonego kodu w polu edycyjnym „Code (Kod)"

UWAGA: Proszę zwrócić uwagę na fakt, że nazwa funkcji lub procedury aktualnie edytowanego kodu jest oznaczona szarym kolorem. Nazwa ta jest widoczna na liście **Templates** (Szablony).

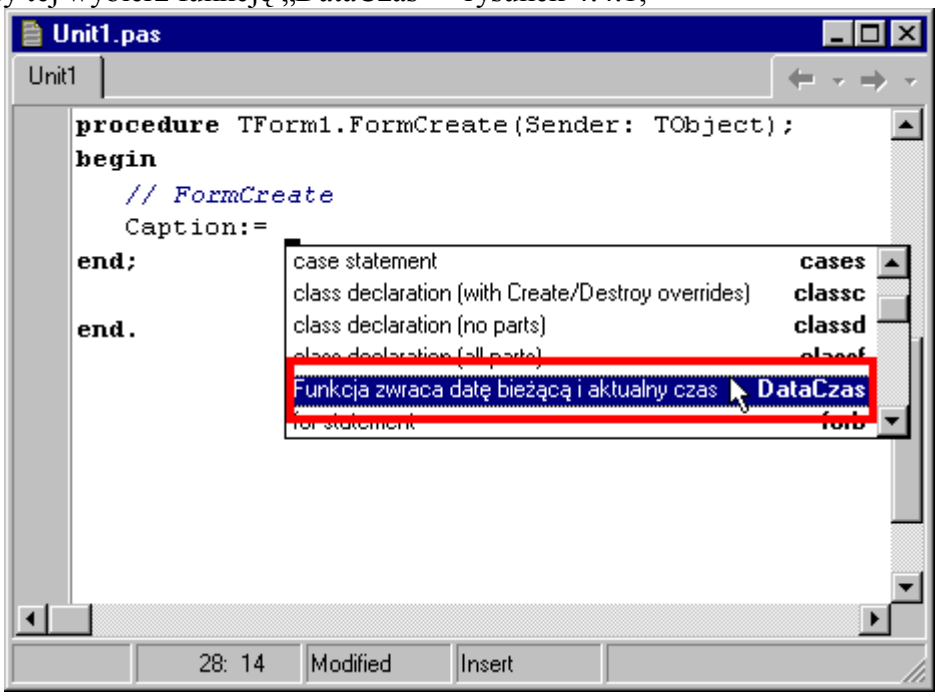
- Po wprowadzeniu kodu dla funkcji „DataCzas" kliknij na klawisz **OK**, w celu zapisania dokonanych zmian.

Ćwiczenie 4.4. Wykorzystanie szablonu kodu

Wykorzystaj kod wprowadzony do szablonu kodu z ćwiczenia 4.3.

Sposób wykonania:

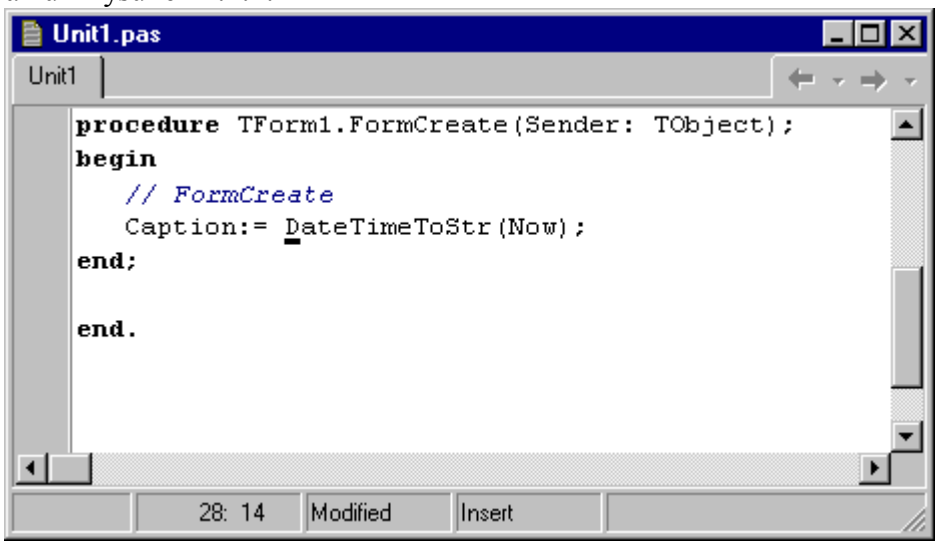
- Będąc w edytorze kodu kliknij kombinację klawiszy **CTRL+J**, co spowoduje wyświetlenie listy szablonów kodu;
- Z listy tej wybierz funkcję „DataCzas" – rysunek 4.4.1;



Rysunek 4.4.1. Widok wybranej funkcji „DataCzas" z listy szablonów kodu

UWAGA: w celu wybrania jakiegoś elementu z listy należy poruszać się strzałkami klawiszowymi góra i dół. Jest również możliwość wciśnięcia klawisza z literą rozpoczynającą nazwę funkcji (w tym przykładzie będzie to klawisz z literą D), co spowoduje przejście do pierwszej funkcji, której nazwa rozpoczyna się od litery wciśniętego klawisza.

- Po wybraniu funkcji, kliknij na niej dwukrotnie lewym klawiszem myszy (lub naciśnij klawisz ENTER), co spowoduje wprowadzenie wybranej funkcji do pisanego programu – rysunek 4.4.2.



Rysunek 4.4.2. Widok wprowadzonej funkcji do pisanego programu

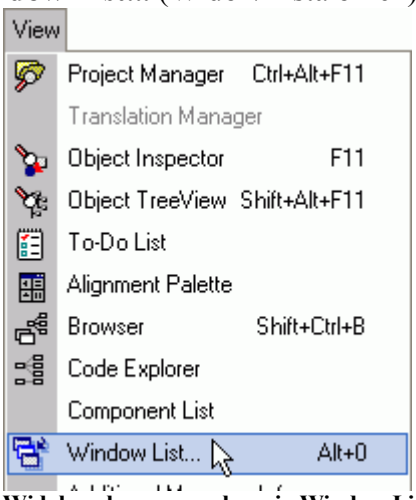
Ćwiczenie 4.5. Lista okien

Przejdź do modułu (pliku z kodem) Unit1 wykorzystując listę okien.

Opis:
Lista okien umożliwia szybkie przenoszenie wybranej listy okna na wierzch wszystkich wyświetlonych okien, bez żmudnego szukania lub przełączania się między tymi oknami.

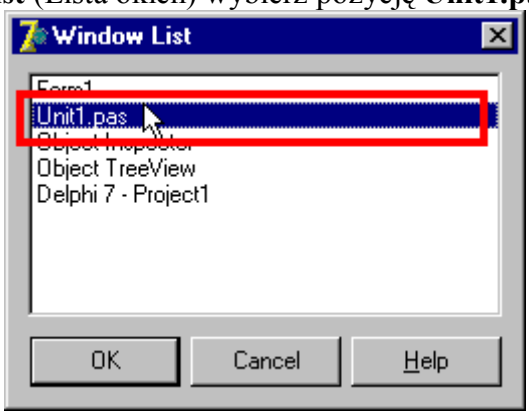
Sposób wykonania:

- Wybierz menu **View/Window List...** (Widok/Lista okien) – rysunek 4.5.1;



Rysunek 4.5.1. Widok wybranego polecenia Window List (Lista okien)

- W oknie **Window List** (Lista okien) wybierz pozycję **Unit1.pas** – rysunek 4.5.2;



Rysunek 4.5.2. Widok wybranej pozycji Unit1.pas

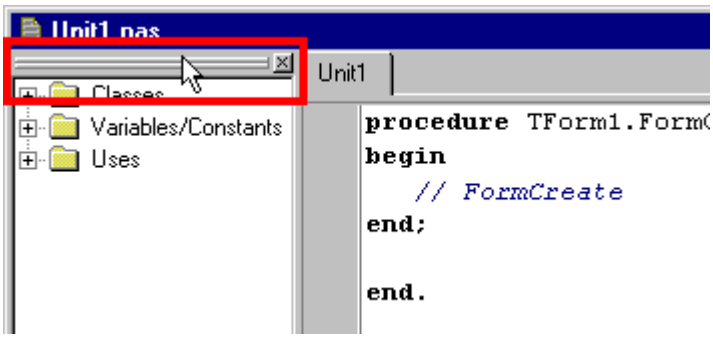
- Po wybraniu kliknij na klawisz **OK** (lub naciśnij klawisz ENTER), co spowoduje przejście do okna modułu **Unit1**.

Ćwiczenie 4.6. Zmiana układu edytora kodu

Przesuń panel Code Explorer (Eksplorator kodu) z lewej strony na prawą stronę edytora kodu.

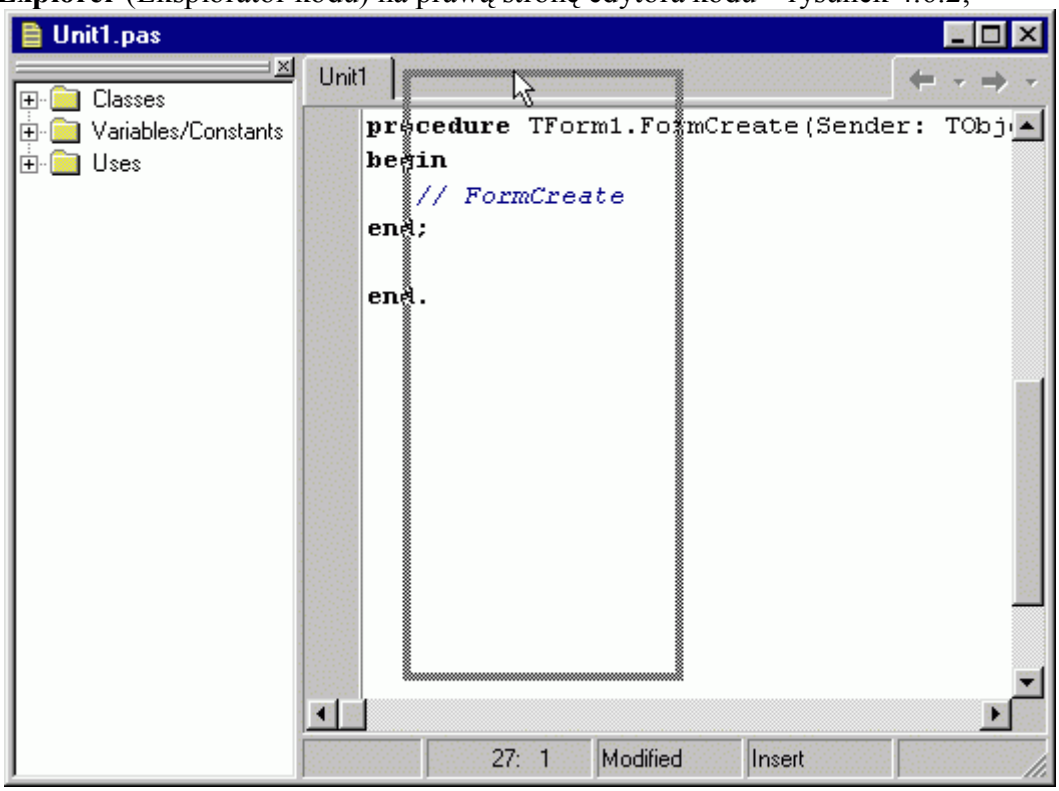
Sposób wykonania:

- Najedź kursorem myszy na dwie linie poziome panelu **Code Explorer** (Eksplorator kodu) – rysunek 4.6.1;



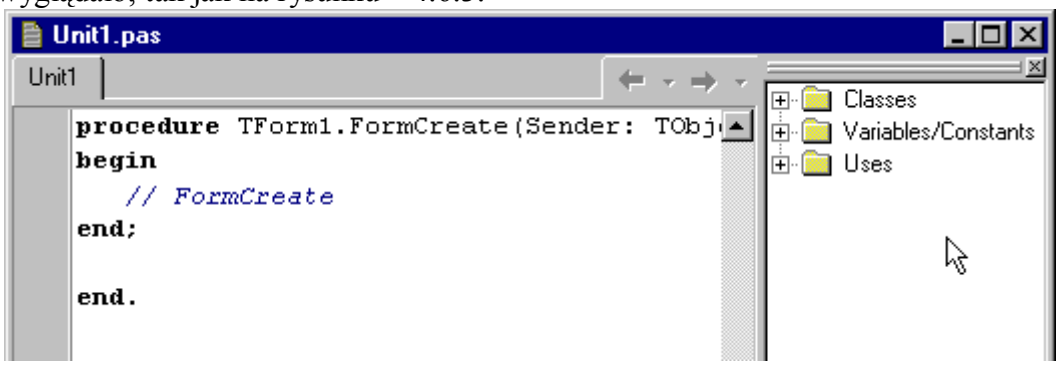
Rysunek 4.6.1. Widok wskazanych dwóch linii poziomych

- Wciśnij lewy klawisz myszy i trzymając go wciśnięty przeciągnij panel **Code Explorer** (Eksplorator kodu) na prawą stronę edytora kodu – rysunek 4.6.2;



Rysunek 4.6.2. Widok przesuwanego panelu

- Po przesunięciu panelu **Code Explorer** (Eksplorator kodu) okno edytora kodu będzie wyglądało, tak jak na rysunku – 4.6.3.



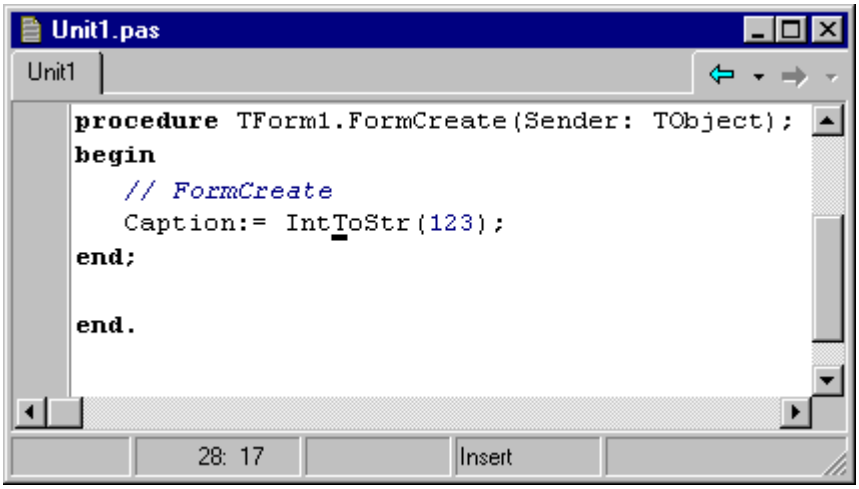
Rysunek 4.6.3. Wygląd okna edytora kodu po przesunięciu panelu Code Explorer (Eksplorator kodu) na prawą stronę

Ćwiczenie 4.7. Korzystanie z pomocy w trakcie pisania programu

Wyświetl pomoc dla funkcji IntToStr() nie korzystając z menu Help(Pomoc).

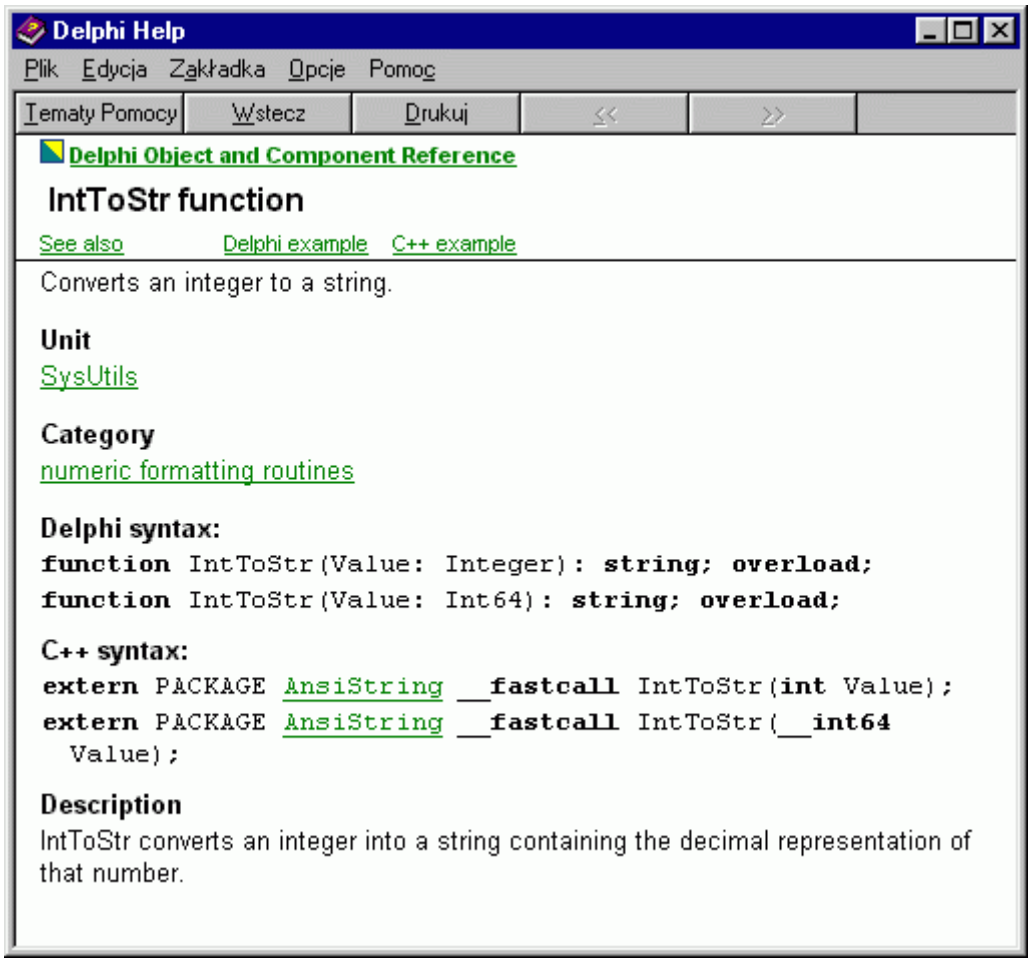
Sposób wykonania:

- Kliknij dwukrotnie lewym klawiszem myszy na formie;
- W wygenerowanym zdarzeniu **OnCreate** (w kodzie programu ma nazwę **FormCreate**) wpisz fragment kodu, np. „Caption:= IntToStr(123);";
- Migający kursor przesuń (np. klawiszami strzałkowymi) pod literę „T" – rysunek 4.7.1;



Rysunek 4.7.1. Widok kursora znajdującego się pod literą „T"

- Naciśnij kombinację klawiszy **CTRL+F1**, co spowoduje wyświetlenie informacji na temat funkcji zaznaczonej migającym kursorem – rysunek 4.7.2.



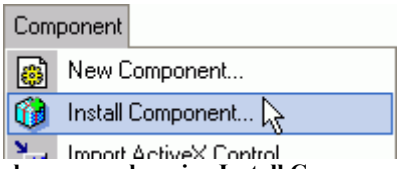
Rysunek 4.7.2. Widok informacji na temat funkcji zaznaczonej migającym kursorem

Ćwiczenie 4.8. Instalacja komponentu

Zainstaluj dowolny komponent, np. COMPLETINGCOMBOBOX.PAS.

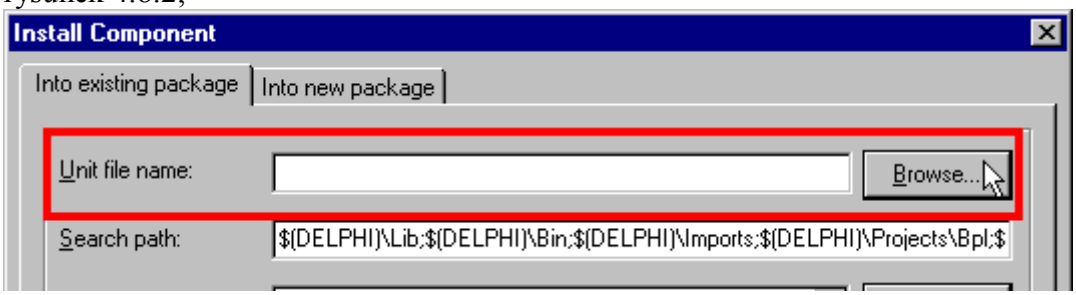
Sposób wykonania:

- Wybierz menu **Component/Install Component…** (Komponent/Instaluj komponent) – rysunek 4.8.1;



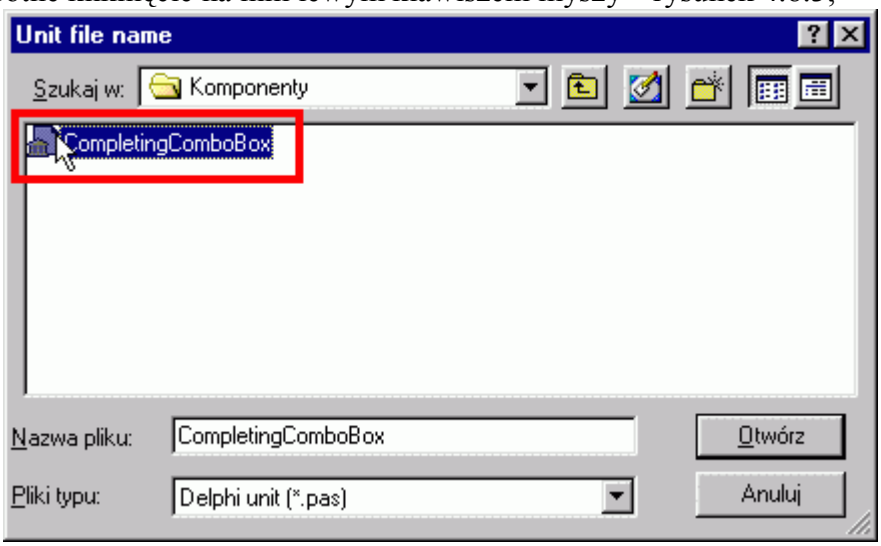
Rysunek 4.8.1. Widok wybranego polecenia „Install Component (Instaluj komponent)”

- W oknie **Install Component** (Instaluj komponent) kliknij na znajdujący się z prawej strony pola edycyjnego **Unit file name** (Nazwa pliku) klawisz **Browse** (Przeglądaj) – rysunek 4.8.2;



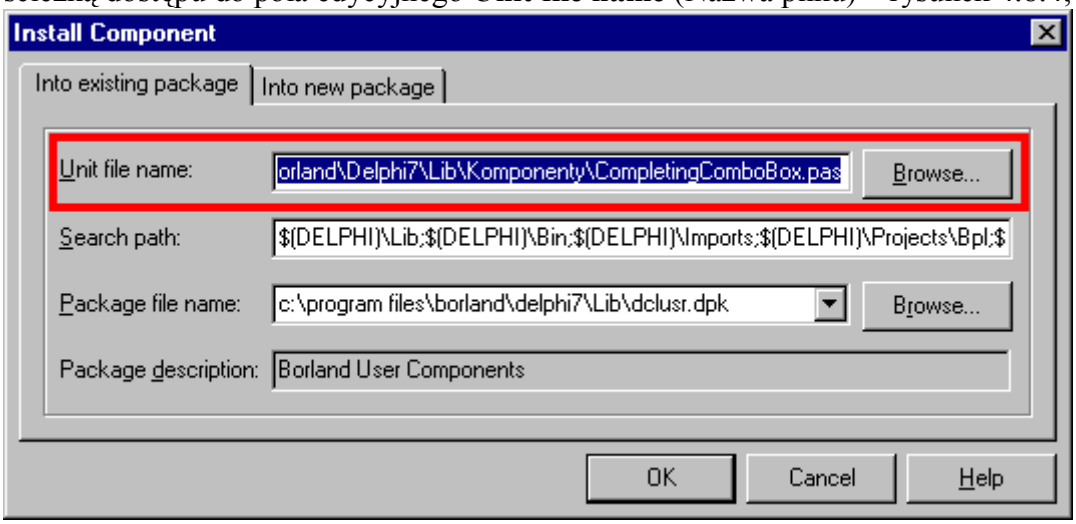
Rysunek 4.8.2. Widok wskazanego klawisza „Browse (Przeglądaj)”

- W oknie **Unit file name** (Nazwa pliku) wybierz katalog, w którym znajduje się komponent (w tym przykładzie komponent znajduje się w katalogu C:\PROGRAM FILES\BORLAND\DELPHI7\LIB\KOMPONENTY), korzystając z listy rozwijanej **Szukaj w** (umożliwiającej znalezienie katalogu);
- Po wybraniu katalogu, wybierz komponent, który chcesz zainstalować, poprzez jednokrotne kliknięcie na nim lewym klawiszem myszy – rysunek 4.8.3;



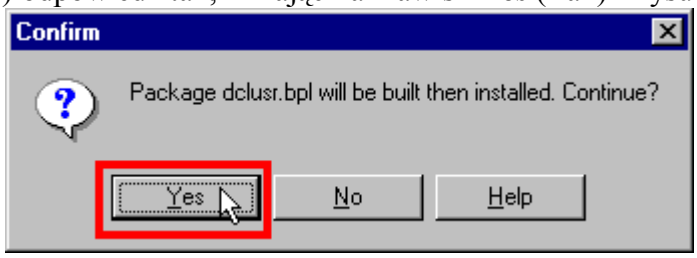
Rysunek 4.8.3. Widok zaznaczonego pliku

- Kliknij na klawisz **Otwórz**, co spowoduje wprowadzenie nazwy komponentu wraz ze ścieżką dostępu do pola edycyjnego **Unit file name** (Nazwa pliku) – rysunek 4.8.4;



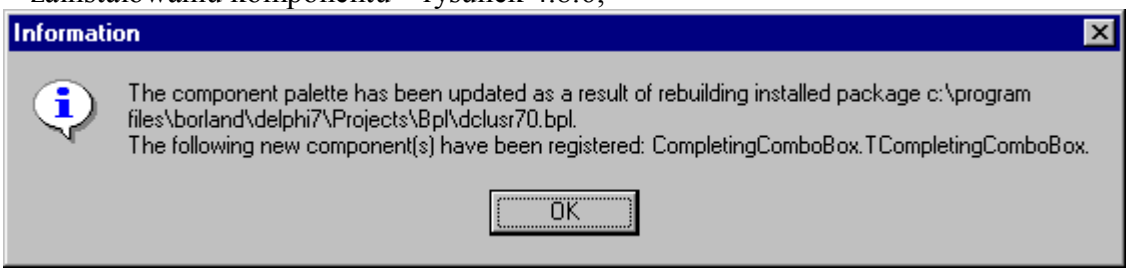
Rysunek 4.8.4. Widok nazwy komponentu wraz ze ścieżką dostępu w polu edycyjnym Unit file name

- Będąc w oknie **Install Component** (Instaluj komponent) kliknij na klawisz **OK**;
- Na pytanie **Package dclusr.bpl will be built then installed. Continue?** (Pakiet dclusr.bpl będzie ponownie kompilowany, co spowoduje zainstalowanie. Czy kontynuować?) odpowiedz tak, klikając na klawisz **Yes** (Tak) – rysunek 4.8.5;



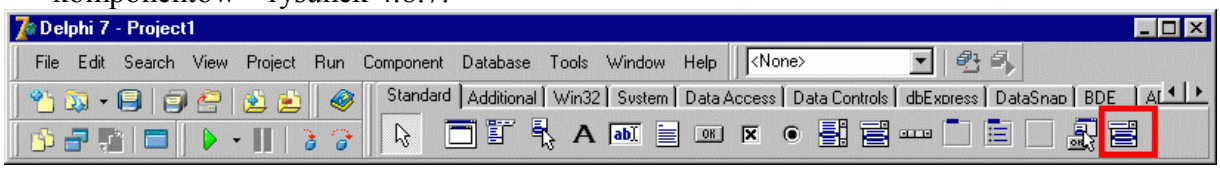
Rysunek 4.8.5. Widok wskazanego klawisza „Yes (Tak)”

- Po pomyślnym zakończeniu operacji, zostanie wyświetlone okno z informacją o zainstalowaniu komponentu – rysunek 4.8.6;



Rysunek 4.8.6. Widok informacji o pomyślnym zainstalowaniu komponentu

- Komponent został pomyślnie zainstalowany i znajduje się na karcie **Standard** palety komponentów – rysunek 4.8.7.



Rysunek 4.8.7. Widok zainstalowanego komponentu

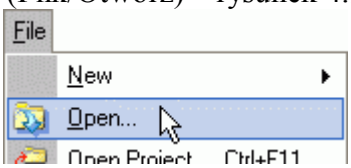
Od tej chwili można już korzystać z nowo zainstalowanego komponentu przy projektowaniu programu.

Ćwiczenie 4.9. Szukanie ciągu znaków

Wczytaj dowolny program (projekt) i znajdź w kodzie programu wyrazy EDIT2.TEXT.

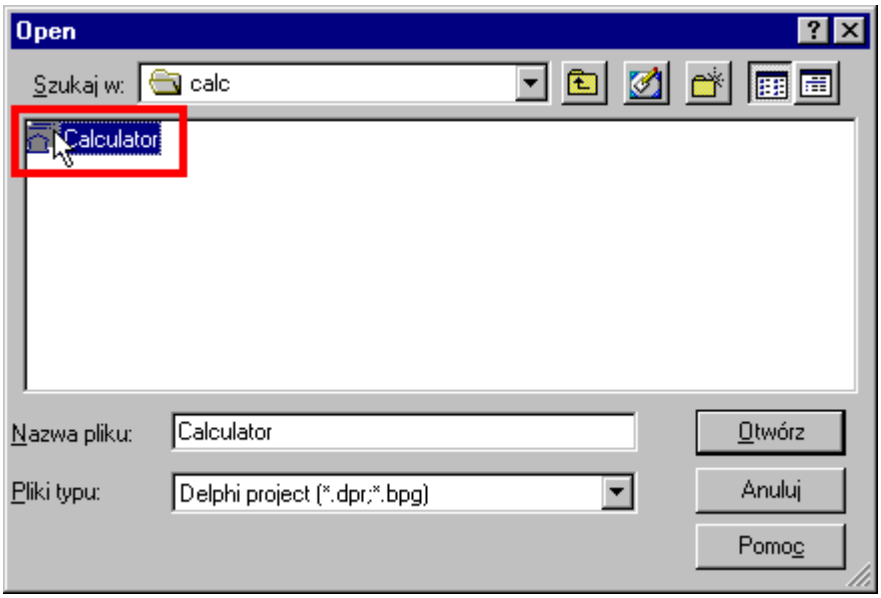
Sposób wykonania:

- Wybierz menu **File/Open...** (Plik/Otwórz) – rysunek 4.9.1;



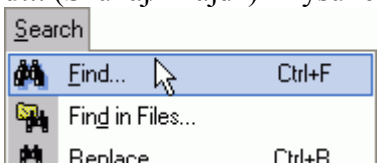
Rysunek 4.9.1. Widok wybranego polecenia „Open (Otwórz)”

- W oknie **Open** (Otwórz) kliknij dwukrotnie lewym klawiszem myszy na wybranym pliku, np. CALCULATOR.DPR – rysunek 4.9.2;



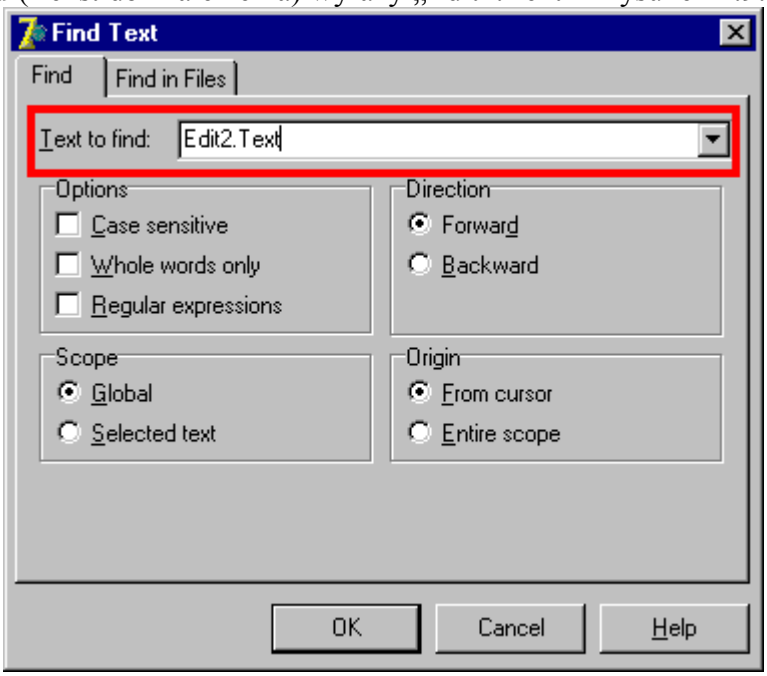
Rysunek 4.9.2. Widok wskazanego pliku CALCULATOR.DPR

- Wybierz menu **Search/Find...** (Szukaj/Znajdź) – rysunek 4.9.3;



Rysunek 4.9.3. Widok wybranego polecenia Find (Znajdź)

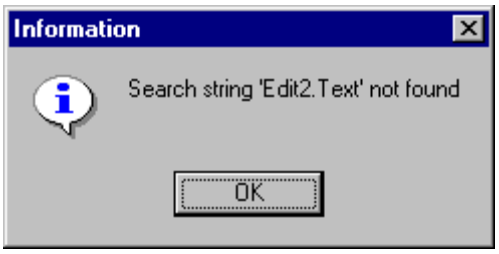
- W oknie **Find Text** (Znajdź tekst) na zakładce **Find** (Znajdź) wpisz w polu edycyjnym **Text to find** (Tekst do znalezienia) wyrazy „Edit2.Text” – rysunek 4.9.4;



Rysunek 4.9.4. Widok wpisanych wyrazów w polu edycyjnym „Text to find (Tekst do znalezienia)”

- Następnie kliknij na klawisz **OK**;
- W celu ponownego szukania tego samego wyrazu naciśnij klawisz **F3**.

Jeżeli szukany wyraz nie istnieje, to zostanie wyświetlone okno informujące o tym fakcie **Search string „Edit2.Text” not found** (Szukany ciąg znaków „Edit2.Text” nie został znaleziony) – rysunek 4.9.5.



Rysunek 4.9.5. Widok okna informującego, że szukany wyraz nie istnieje

Naciśnięcie klawisza **OK** powoduje przejście do kodu programu.

Ćwiczenie 4.10. Uruchomienie projektu w środowisku Delphi

Uruchom projekt automatycznie utworzony przez Delphi w momencie startu.

Opis:
W momencie uruchomienia programu Delphi, automatycznie tworzony jest projekt. Projekt składa się z jednej formatki (domyślna nazwa **Form1** – na niej można układać komponenty) i jednego modułu (domyślna nazwa **Unit1** – w nim można pisać kod programu). Projekt ten można dowolnie rozbudować. Projekt utworzony przy uruchomieniu Delphi nie posiada żadnego komponentu, a jego kod jest niewielki. Po uruchomieniu takiego projektu wyświetlona zostanie tylko pusta formatka.

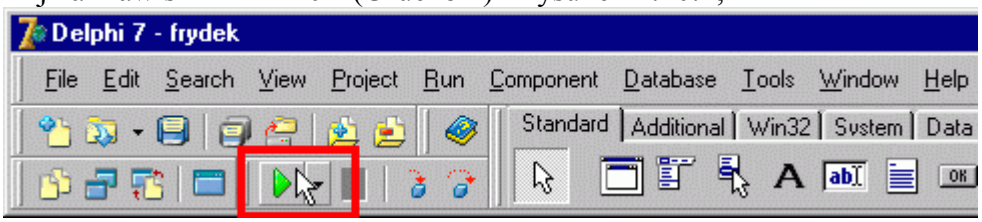
Sposób 1:

- Wybierz menu **Run/Run** (Uruchom/Uruchom) – rysunek 4.10.1;

Rysunek 4.10.1. Widok wybranego polecenia Run (Uruchom)

Sposób 2:

- Kliknij na klawisz **Run** (Uruchom) – rysunek 4.10.2;



Rysunek 4.10.2. Widok wskazanego klawisza Run (Uruchom)

Po wykonaniu czynności jednym z wyżej opisanych sposobów, nastąpi uruchomienie programu.

Ćwiczenie 4.11. Kompilacja całego projektu

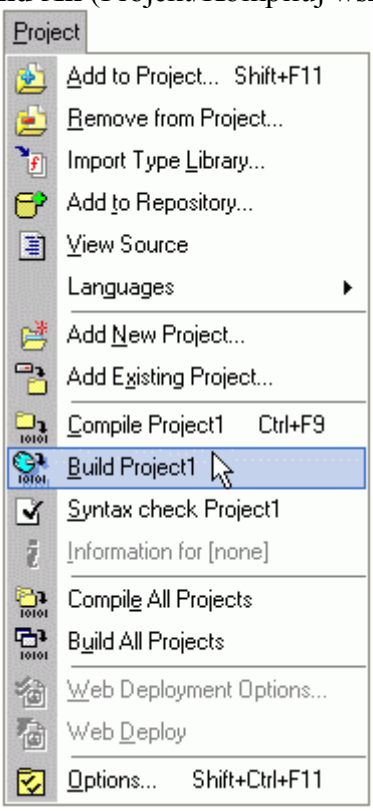
Wykonaj kompilację całego projektu.

Opis:

Kompilacja polega na tłumaczeniu programu (napisanego w języku programowania zrozumiałym dla człowieka, np. Pascal, C++, Basic) z postaci źródłowej na kod wynikowy (kod maszynowy). Dopiero po wykonaniu kompilacji programu, można taki program uruchomić poza środowiskiem programowania w jakim był tworzony.

Sposób wykonania:

- Wybierz menu **Project/Build All** (Projekt/Kompiluj wszystko) – rysunek 4.11.1;



Rysunek 4.11.1. Widok wybranego polecenia Build Project1 (Kompiluj Project1)

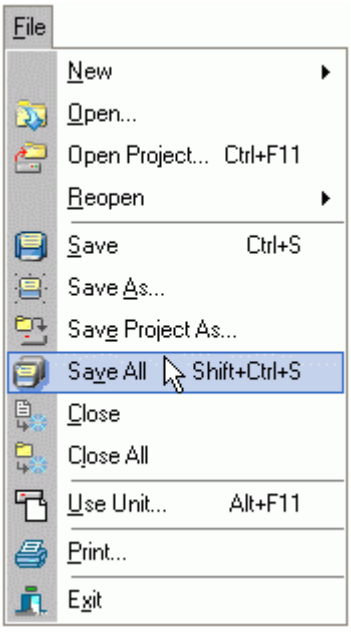
Po wykonaniu tej czynności projekt zostanie skompilowany.

Ćwiczenie 4.12. Zapisywanie projektu

Zapisz projekt w katalogu DELPHI, który znajduje się na dysku C: w katalogu głównym. Plik modułu ma być zapisany pod nazwą np. UPROG, natomiast plik projektu ma mieć nazwę np. PIERWSZY.

Sposób 1:

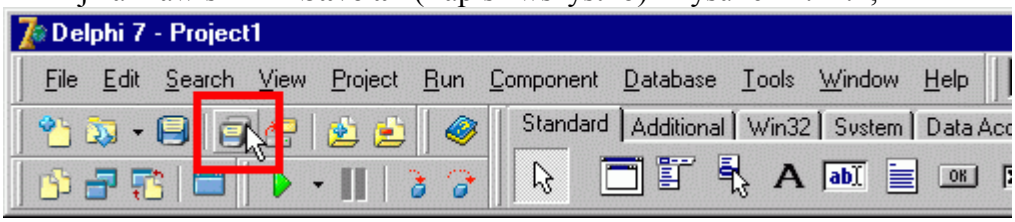
- Wybierz z menu **File/Save all** (Plik/Zapisz wszystko) – rysunek 4.12.1;



Rysunek 4.12.1. Widok wybranego polecenia Save all (Zapisz wszystko)

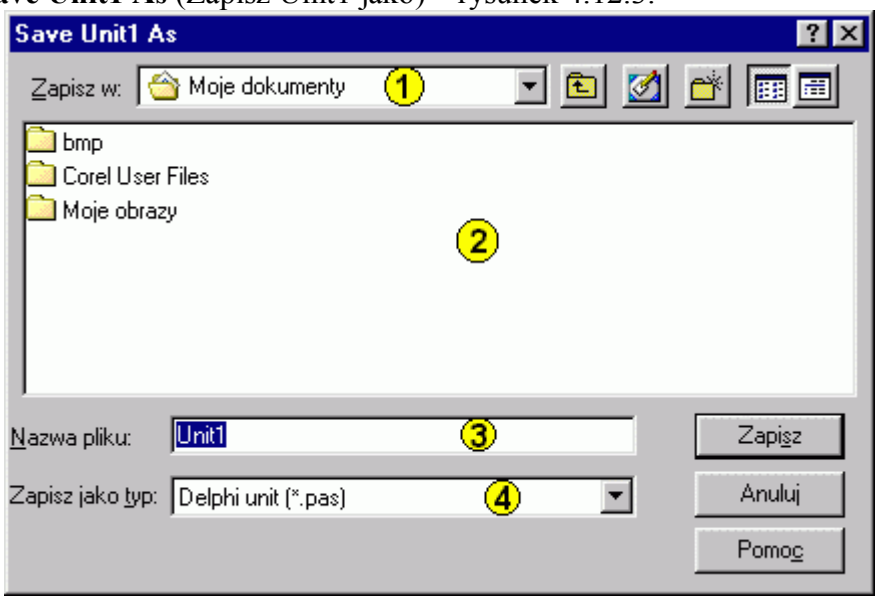
Sposób 2:

- Kliknij na klawisz **Save all** (Zapisz wszystko) – rysunek 4.12.2;



Rysunek 4.12.2. Widok wskazanego klawisza kursorem myszy

Po wykonaniu czynności jednym z wyżej opisanych sposobów zostanie wyświetlone okno dialogowe **Save Unit1 As** (Zapisz Unit1 jako) – rysunek 4.12.3.



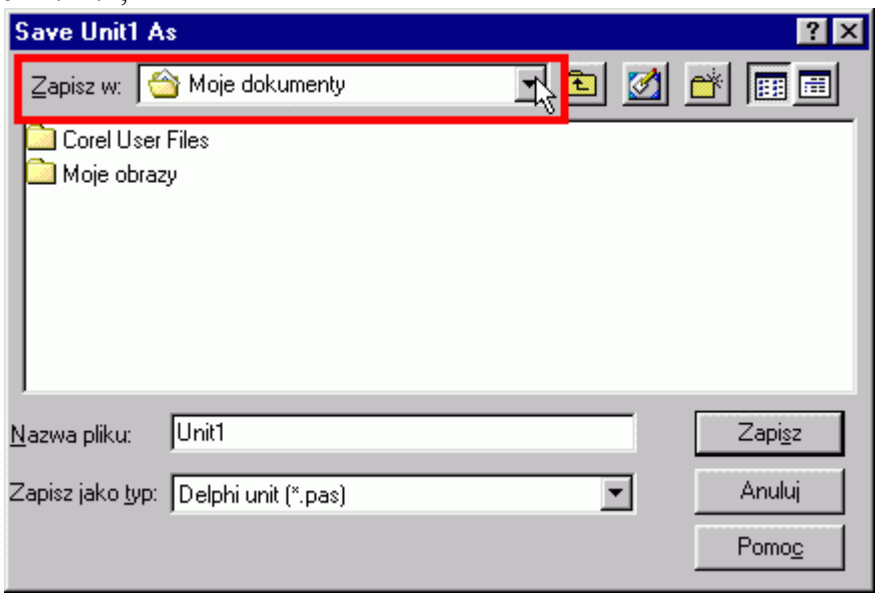
Rysunek 4.12.3. Widok okna dialogowego Save Unit1 As (Zapisz Unit1 jako)

Opis okna:

1) Lista rozwijana **Zapisz w**, umożliwia wyszukanie dysku lub/i katalogu, w którym ma być zapisany plik;
2) **Lista katalogów i plików** (zajmuje największą część okna dialogowego), umożliwia wyświetlenie wszystkich plików znajdujących się w danym katalogu;
3) Pole **Nazwa pliku**, umożliwia podanie nazwy, pod którą zostanie zapisany plik (domyślną nazwą proponowaną przez Delphi jest **Unit1**);
4) Lista rozwijana **Zapisz jako typ**, umożliwia wybranie typu (formatu), w jakim ma zostać zapisany plik. Od wybranego typu, zależy jakie pliki będą wyświetlane w polu **Lista katalogów i plików**.

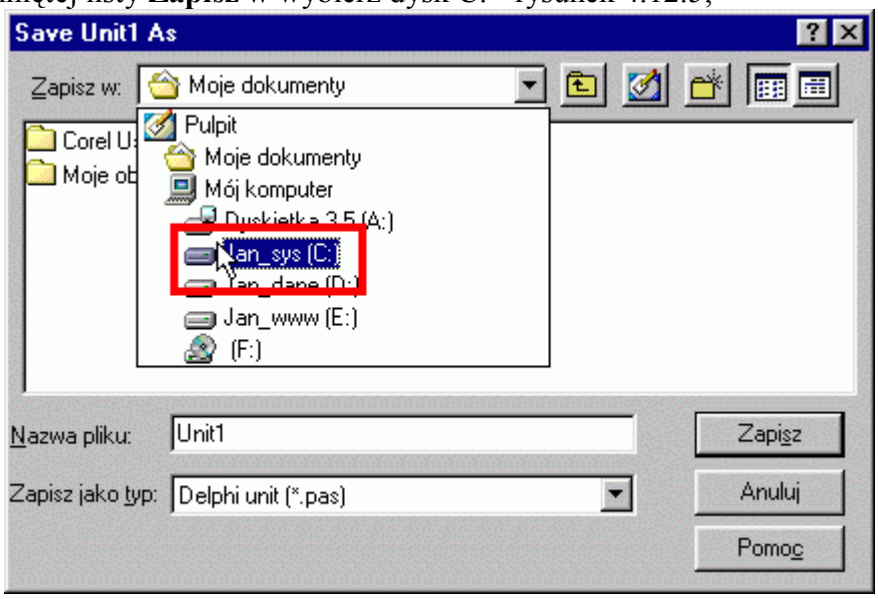
UWAGA: Przy zapisywaniu projektu, program Delphi sam określa typ (format) w jakim zostanie zapisany plik. Tak więc nie ma potrzeby dokonywania zmiany typu (formatu).

- Rozwiń listę **Zapisz w**, klikając na klawisz (znajduje się on, z prawej strony listy) – rysunek 4.12.4;



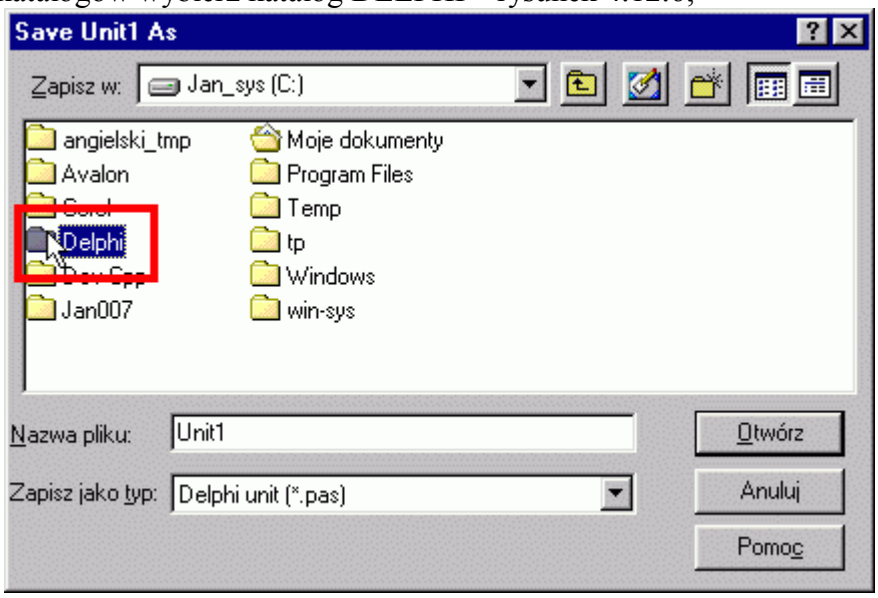
Rysunek 4.12.4. Widok wskazanej strzałki umożliwiającej rozwinięcie listy

- Z rozwiniętej listy **Zapisz w** wybierz dysk C: - rysunek 4.12.5;



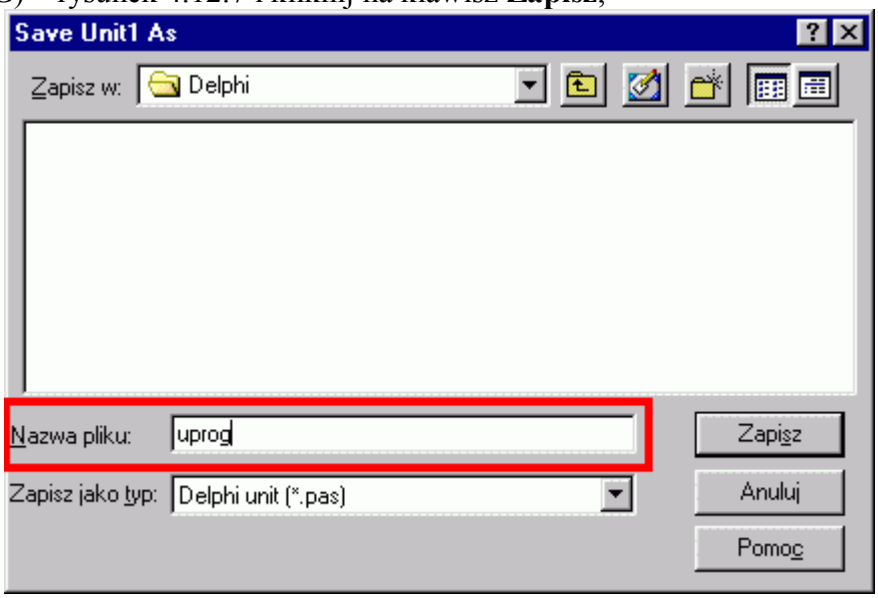
Rysunek 4.12.5. Widok wybranego dysku C:

- Po wybraniu dysku w oknie **lista katalogów i plików** zostanie wyświetlona zawartość katalogu głównego dysku C:;
- Z listy katalogów wybierz katalog DELPHI – rysunek 4.12.6;



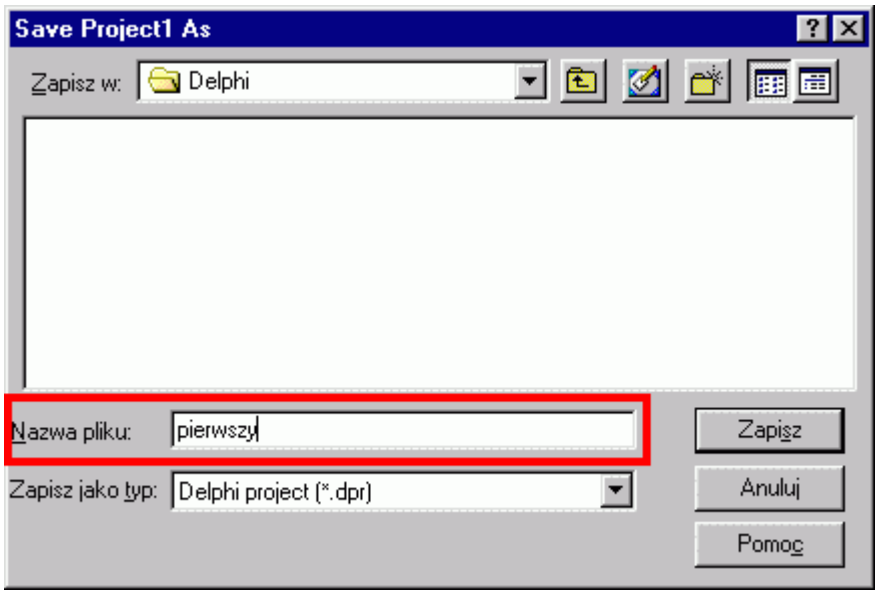
Rysunek 4.12.6. Widok wskazanego katalogu DELPHI na dysku C:

- Kliknij dwukrotnie lewym klawiszem myszy na wybranym katalogu, co spowoduje wejście do niego;
- W polu **Nazwa pliku** wpisz nazwę, pod jaką ma zostać zapisany plik-moduł (np. UPROG) – rysunek 4.12.7 i kliknij na klawisz **Zapisz**;



Rysunek 4.12.7. Widok wprowadzonej nazwy pliku-modułu „uprog”

- W oknie dialogowym **Save Project1 As** (Zapisz Project1 jako) wpisz w polu **Nazwa pliku** nazwę, pod jaką ma zostać zapisany plik-projektu na dysku (np. PIERWSZY) – rysunek 4.12.8;



Rysunek 4.12.8. Widok wpisanej nazwy pliku-projektu „pierwszy"

- Kliknij na klawisz **Zapisz**.

UWAGA: Jeżeli projekt składa się z większej ilości modułów, to przy zapisywaniu Delphi będzie prosił o podanie nazwy poszczególnych modułów. Pod podanymi nazwami moduły będą zapisywane w postaci plików na dysku.

5. Ćwiczenia z podstaw projektowania programu

Poniżej przedstawione są ćwiczenia dotyczące podstaw projektowania programu w środowisku Delphi 7. Dla jaśniejszego zrozumienia ćwiczeń będę je przedstawiał w oparciu o jeden komponent (np. komponent BUTTON).

Ćwiczenie 5.1. Tworzenie nowego projektu i dodanie nowej formy

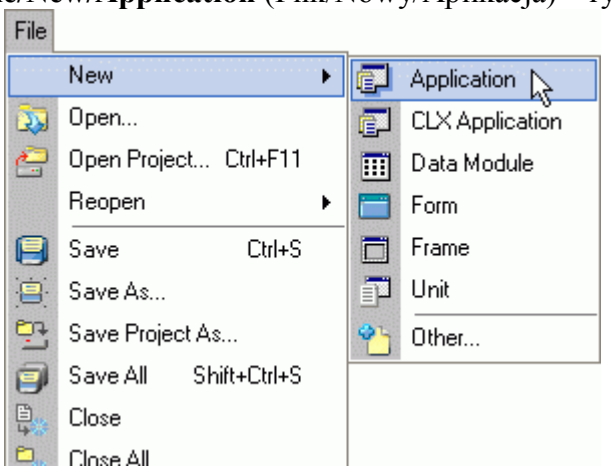
Pokaż jak uruchomić nowy projekt oraz jak dodać do projektu nową formę.

Opis:
W momencie uruchomienia Delphi, automatycznie tworzony jest nowy projekt.
W skład projektu wchodzi: formatka (domyślna nazwa formatki **Form1**), moduł (nazwa domyślna **Unit1.pas** – w nim znajduje się kod programu), główny plik projektu (domyślna nazwa **Project1.dpr**) oraz plik z danymi do formatki (nazwa pliku formatki jest taka sama jak

nazwa modułu, z wyjątkiem rozszerzenia, np. **Unit1.dfm** – zawiera dane o komponentach umieszczonych na formatce).

Sposób stworzenia nowego projektu:

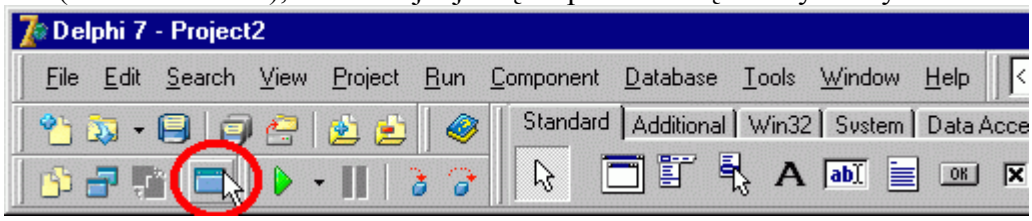
- Wybierz menu **File/New/Application** (Plik/Nowy/Aplikacja) – rysunek 5.1.1;



Rysunek 5.1.1. Widok wybranego polecenia Application (Aplikacja)

Sposób dodania nowej formy do projektu:

- Wybierz menu **File/New/Form** (Plik/Nowy/Formatka) lub kliknij na ikonę **New form** (Nowa formatka), która znajduje się na pasku narzędziowym – rysunek 5.1.2;



Rysunek 5.1.2. Widok wskazanej ikony New form (Nowa formatka) na pasku narzędziowym

Dodana formatka będzie posiadała nazwę domyślną, np. **Form2**. Natomiast plik modułu będzie posiadał nazwę domyślną, np. **Unit2**. Nazwy te można zmieniać m.in. w momencie zapisywania projektu na dysk twardy.

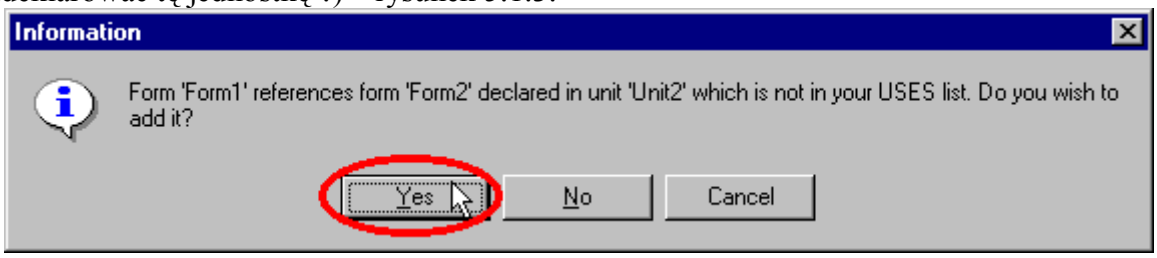
W celu wywołania drugiej formatki (np. Form2) z formatki głównej (Form1), należy umieścić linię „Form2.ShowModal;" w funkcji wywołującej drugą formę. Na przykład:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   Form2.ShowModal;
end;
```

Naciśnięcie na klawisz „Button1" spowoduje wywołanie drugiej formatki (nazwa domyślna Form2).

UWAGA:
W momencie kompilowania programu, Delphi wyświetli komunikat **Form 'Form1' references from 'Form2' declared in unit 'Unit2' which is not in your USES list. Do you wish to add it?** (Forma 'Form1' wywołuje deklaracje z 'Form2' znajdującą się w jednostce

'Unit2', która nie jest zadeklarowana w liście po słowie USES. Czy życzysz sobie zadeklarować tę jednostkę ?) – rysunek 5.1.3.



Rysunek 5.1.3. Widok okna informacyjnego

Kliknij na klawisz z napisem **Yes** (Tak), przez co wyrażasz zgodę na zadeklarowanie jednostki „Unit2". Przykład zadeklarowania jednostki przedstawiony jest poniżej:

implementation

uses Unit2;

*{$R *.DFM}*

Zwróć uwagę na fakt, że deklaracja nowych formatek znajduje się po słowie **implementation**, a przed dyrektywą *{$R *.DFM}* i jakąkolwiek inną funkcją lub procedurą.

Ćwiczenie 5.2. Generowanie zdarzenia

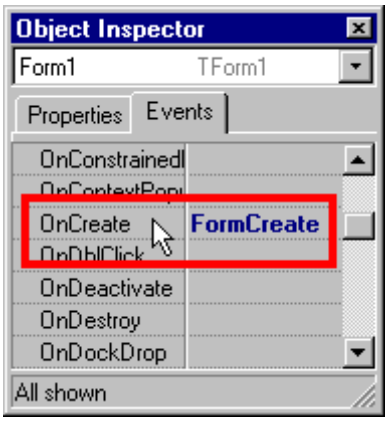
Wygeneruj zdarzenie OnCreate dla formy (nazwa domyślna formatki Form1).

Opis: opis wybranych zdarzeń znajduje się w podrozdziale 3.9.

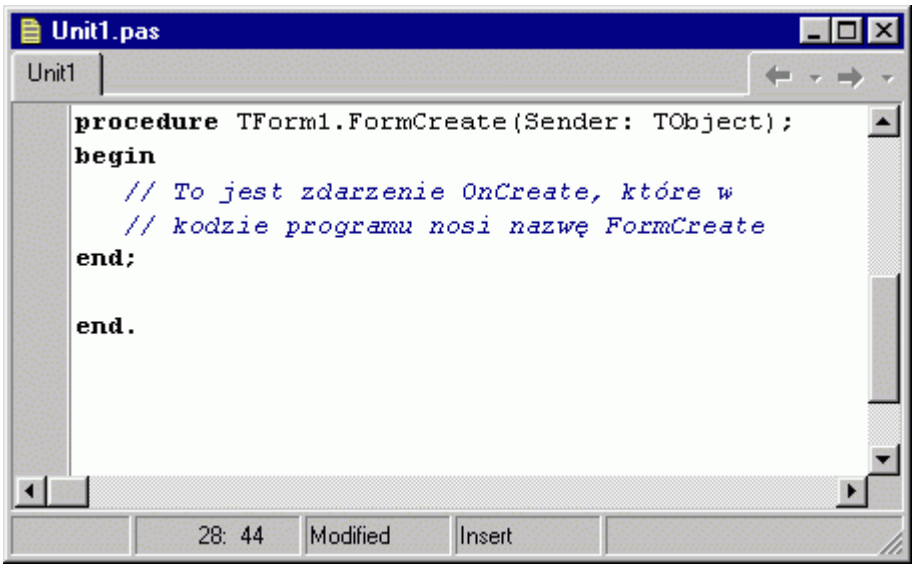
Sposób wykonania:

- Kliknij dwukrotnie lewym klawiszem myszy na formie (domyślna nazwa formy to **Form1**);

UWAGA: Zwróć uwagę na to, że zdarzenie **OnCreate** (taka nazwa widnieje w oknie Inspektora obiektów – rysunek 5.2.1.) w kodzie programu nosi nazwę **FormCreate** – rysunek 5.2.2.



Rysunek 5.2.1. Widok nazwy zdarzenia OnCreate w oknie Inspektora Obiektów, a obok nazwa zdarzenia istniejąca w kodzie programu pod nazwą FormCreate

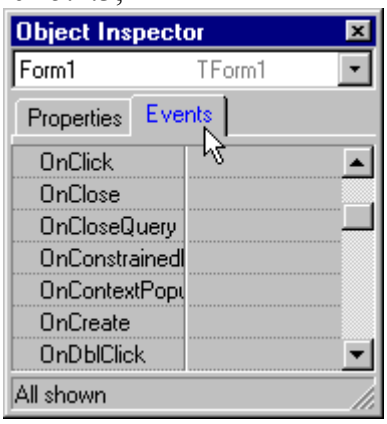


Rysunek 5.2.2. Widok wygenerowanego zdarzenia OnCreate, który w kodzie programu ma nazwę FormCreate

Instrukcje (rozkazy) zawarte w zdarzeniu **OnCreate** (nazwa w kodzie programu **FormCreate**) wykonywane są w momencie uruchomienia programu.

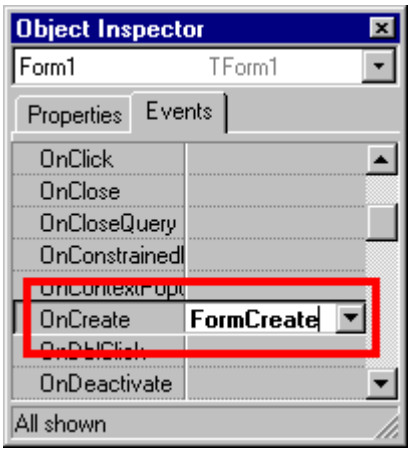
Innym sposobem generowania zdarzeń, jest skorzystanie z **Object Inspector'a** (Inspektora obiektów). Poniżej są podane kroki jakie należy wykonać, aby zdarzenie zostało wygenerowane:

- Przejdź do okna **Object Inspector'a** (Inspektora obiektów) naciskając klawisz funkcyjny **F11** lub wybierając menu **View/Object Inspector** (Widok/Inspektor obiektów);
- Będąc w oknie **Object Inspector'a** (Inspektora obiektów), przejdź na zakładkę **Events** (Zdarzenia) – rysunek 5.2.3;



Rysunek 5.2.3. Widok listy zdarzeń w oknie Object Inspector (Inspektora obiektów)

- Kliknij dwukrotnie lewym klawiszem myszy z prawej strony napisu **OnCreate** (druga kolumna) - rysunek 5.2.4;



Rysunek 5.2.4. Widok nazwy wygenerowanego zdarzenia OnCreate

Po wygenerowaniu zdarzenia **OnCreate** za pomocą **Object Inspector'a** (Inspektora obiektów) automatycznie następuje przejście do kodu programu.

Ćwiczenie 5.3. Nazwa programu

Nadaj nazwę programowi niezależną od nazwy pliku programu.

Opis:
Nazwa aplikacji (programu), która jest wyświetlana na pasku zadań jest pobierana z nazwy pliku. Nazwę programu możemy określić w trakcie jego tworzenia, dzięki czemu nazwa programu wyświetlana na pasku zadań będzie inna niż nazwa pliku programu.

Określenie nazwy programu - sposób 1:

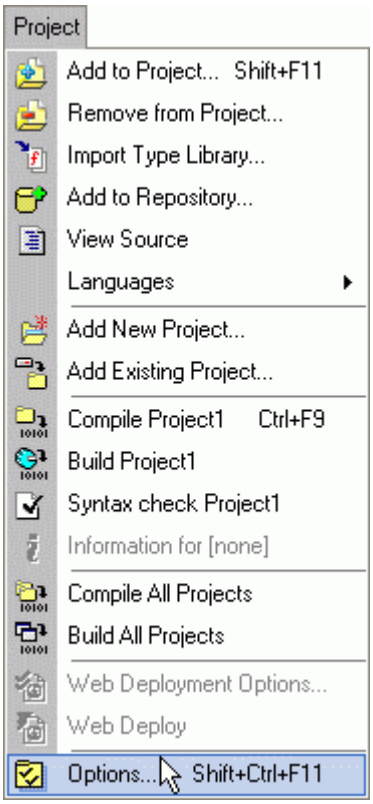
- Kliknij dwukrotnie lewym klawiszem myszy na formie, co spowoduje wygenerowanie zdarzenia **OnCreate** (nazwa w kodzie programu **FormCreate**);
- W wygenerowanym zdarzeniu wpisz linię **Application.Title:= 'Mój program';** - przykład poniżej;

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.Title:= 'Mój program';
end;
```

W momencie uruchomienia programu, zostanie mu przypisana nazwa „Mój program".

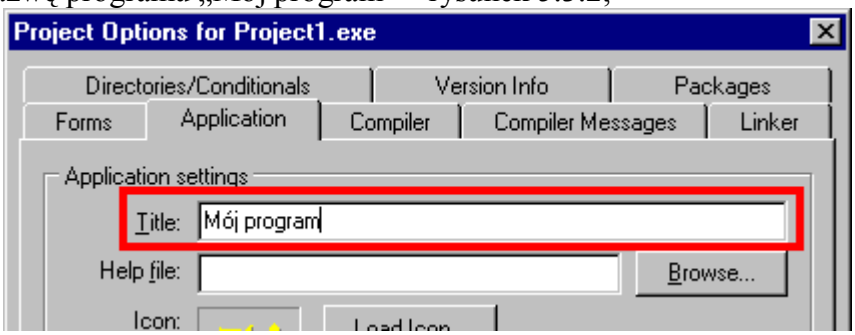
Określenie nazwy programu - sposób 2:

- Wybierz menu **Project/Options...** (Projekt/Opcje) – rysunek 5.3.1



Rysunek 5.3.1. Widok wybranego polecenia Options (Opcje)

- W oknie **Project Options** (Opcje projektu) kliknij na zakładkę **Application** (Aplikacja);
- W sekcji **Application settings** (Ustawienia programu) w polu edycyjnym **Title** (Tytuł) wpisz nazwę programu „Mój program" – rysunek 5.3.2;

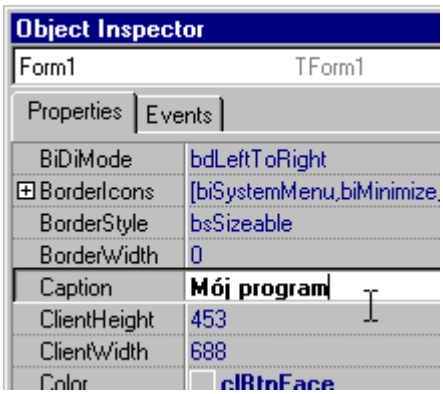


Rysunek 5.3.2. Widok wpisanej nazwy programu

- Kliknij klawisz **OK**, w celu zatwierdzenia dokonanych zmian.

Określenie nazwy formatki – sposób 1:
W celu zmiany domyślnej nazwy formatki (domyślną nazwą jest **Form1**) na inną, należy wykonać następujące kroki:

- Kliknij na formatce;
- Przejdź do okna **Object Inspector'a** (Inspektora obiektów) wybierając menu **View/Object Inspector** lub naciskając klawisz funkcyjny **F11**;
- Będąc na zakładce **Properties** (Właściwości) okna **Object Inspector'a** (Inspektora obiektów) znajdź pozycję listy o nazwie **Caption** (Podpis) przesuwając belką przewijania;
- Z prawej strony napisu **Caption** (Podpis) wpisz nazwę programu „Mój program", który ukaże się na pasku tytułowym formatki – rysunek 5.3.3;



Rysunek 5.3.3. Widok wpisanej nazwy, która będzie widoczna na pasku tytułowym programu

Określenie nazwy formatki – sposób 2:

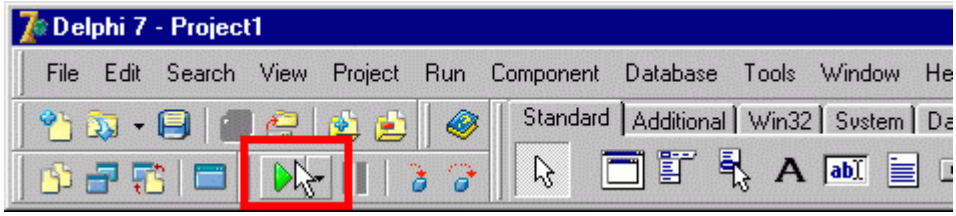
- Kliknij dwukrotnie lewym klawiszem myszy na formie, co spowoduje wygenerowanie zdarzenia **OnCreate** (nazwa w kodzie programu **FormCreate**);
- W wygenerowanym zdarzeniu wpisz linię **Caption:= 'Mój program';** - przykład poniżej;

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Caption:= 'Mój program';
end;
```

W momencie uruchomienia programu, nazwa zostanie wyświetlona na pasku tytułowym formatki.

Po nadaniu nazwy formatce i programowi, uruchom program wykonując następujące czynności:

- Uruchom program naciskając klawisz funkcyjny **F9** lub kliknij na klawisz **Run** (Uruchom), który znajduje się na pasku narzędziowym – rysunek 5.3.4;



Rysunek 5.3.4. Widok wskazanej ikony Run(Uruchom) na pasku narzędziowym

W momencie uruchomienia aplikacji, nazwa programu na pasku zadań będzie taka sama, jak nazwa na pasku tytułowym formatki.

Ćwiczenie 5.4. Ikona programu

Wczytaj ikonę do programu oraz do paska tytułowego.

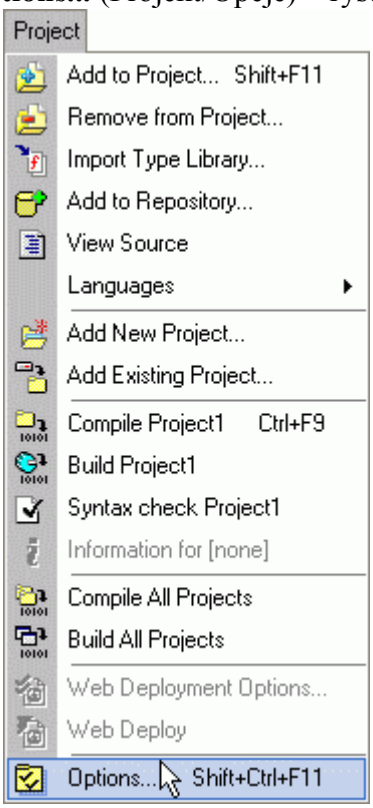
Opis:

Ikona aplikacji (programu) służy do jednoznacznego identyfikowania programu pod względem jego przeznaczenia. Jest to rysunek o rozmiarach 32x32 piksele, który przynajmniej w przybliżeniu powinien mówić o przeznaczeniu programu, np.

- - określa, że program służy np. do przetwarzania dokumentów (plików tekstowych);
- - określa, że program służy do tworzenia np. filmów.

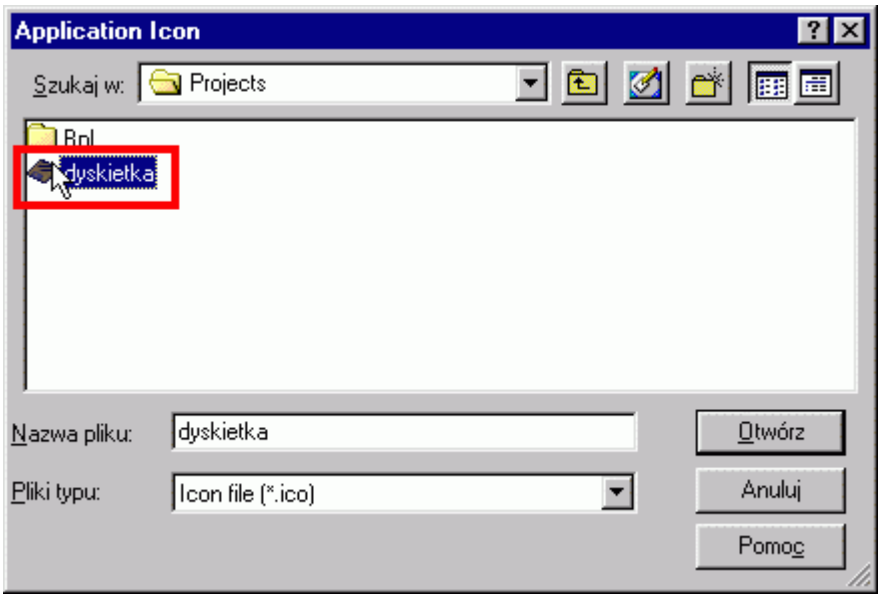
Ustawienie ikony dla programu:

- Wybierz menu **Project/Options...** (Projekt/Opcje) – rysunek 5.4.1



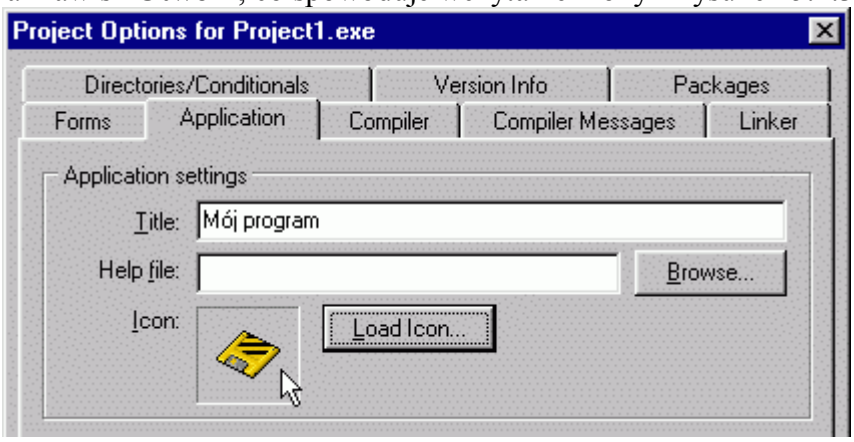
Rysunek 5.4.1. Widok wybranego polecenia Options (Opcje)

- W oknie **Project Options** (Opcje projektu) kliknij na zakładkę **Application** (Aplikacja);
- Kliknij na klawisz **Load Icon** (Wczytaj ikonę);
- W oknie **Application Icon** (Ikona programu) wybierz z listy ikonę – rysunek 5.4.2;



Rysunek 5.4.2. Widok zaznaczonej ikony

- Kliknij na klawisz **Otwórz**, co spowoduje wczytanie ikony – rysunek 5.4.3;

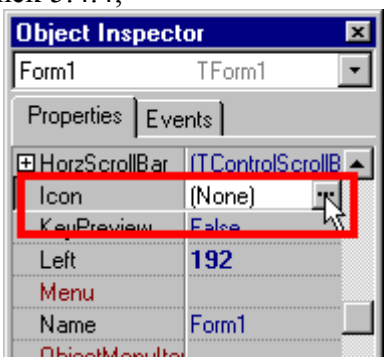


Rysunek 5.4.3. Widok wczytanej ikony

- Kliknij na klawisz **OK**, w celu zatwierdzenia dokonanych zmian.

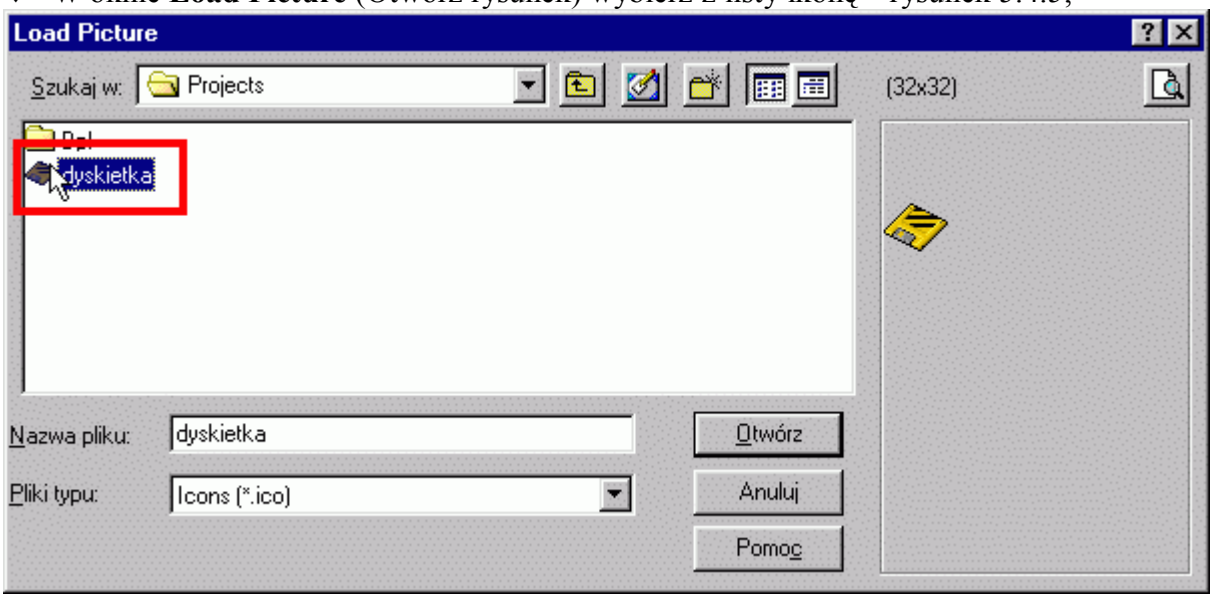
Ustawienie ikony dla formatki (okna programu):

- Przejdź do okna **Object Inspector'a** (Inspektora obiektów) wybierając menu **View/Object Inspector** lub naciskając klawisz funkcyjny **F11**;
- Będąc na zakładce **Properties** (Właściwości) okna **Object Inspector'a** (Inspektora obiektów) znajdź pozycję listy o nazwie **Icon** (Ikona) przesuwając belką przewijania;
- Po znalezieniu tego elementu kliknij na klawisz ..., który znajduje się z prawej strony napisu **Icon** (Ikona) - rysunek 5.4.4;



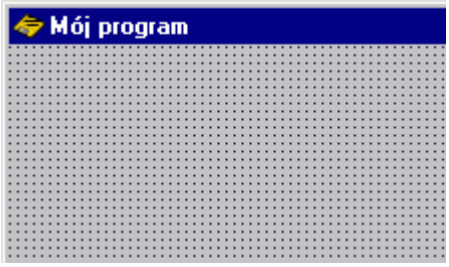
Rysunek 5.4.4. Widok wskazanego klawisza ...

- W oknie **Picture editor** (Edytor rysunku) kliknij na klawisz **Load** (Otwórz);
- W oknie **Load Picture** (Otwórz rysunek) wybierz z listy ikonę – rysunek 5.4.5;



Rysunek 5.4.5. Widok wybranej ikony

- Kliknij na klawisz **Otwórz**, co spowoduje wyświetlenie ikony w oknie **Load Picture** (Otwórz rysunek);
- Kliknij na klawisz **OK**, co spowoduje wstawienie ikony do paska tytułowego – rysunek 5.4.6;



Rysunek 5.4.6. Widok ikony na pasku tytułowym

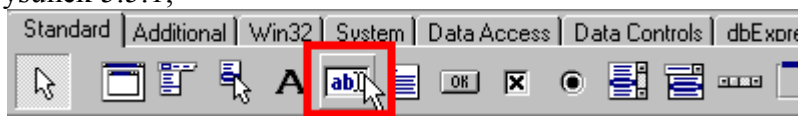
W momencie uruchomienia aplikacji, ikona programu na pasku zadań będzie taka sama, jak ikona na pasku tytułowym formatki.

Ćwiczenie 5.5. Umieszczenie komponentu na formatce

Umieść na formatce dowolny komponent, np. EDIT.

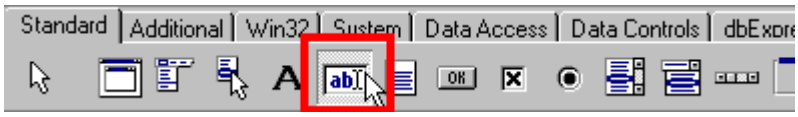
Sposób wykonania:

- Wskaż kursorem myszy komponent **Edit** z palety komponentów (zakładka **Standard**) – rysunek 5.5.1;



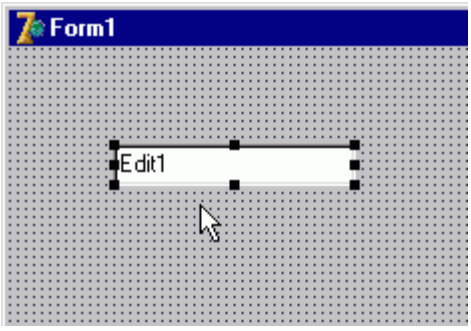
Rysunek 5.5.1. Widok wskazanego komponentu EDIT na zakładce Standard

- Kliknij na ten komponent lewym klawiszem myszy, w celu oznaczenia (wybrany komponent jest wklęsły) – rysunek 5.5.2;



Rysunek 5.5.2. Widok oznaczonego (tzn. wklęsłego) komponentu EDIT

- Kliknij na formatce w miejscu, w którym ma się znaleźć wybrany komponent – rysunek 5.5.3;



Rysunek 5.5.3. Widok umieszczonego na formatce komponentu EDIT

Umieszczony na formatce komponent jest oznaczony przez 8 kwadracików, które umożliwiają zmianę szerokości i wysokości. Komponent umieszczony na formatce można również przesuwać (kliknij jednokrotnie lewym klawiszem myszy na komponent i przytrzymując wciśnięty klawisz myszy przesuń komponent w inne miejsce).

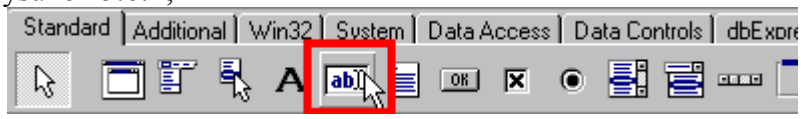
UWAGA: Zwróć uwagę na to, że oznaczony jest tylko jeden komponent. Oznaczenie to, przechodzi na ostatni umieszczony komponent na formatce.

Ćwiczenie 5.6. Umieszczenie kilku tych samych komponentów na formatce

Umieść na formatce kilka tych samych komponentów, np. EDIT.

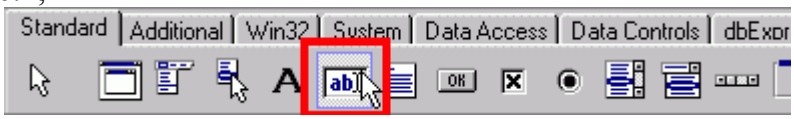
Sposób wykonania:

- Wskaż kursorem myszy komponent **Edit** z palety komponentów (zakładka **Standard**) – rysunek 5.6.1;



Rysunek 5.6.1. Widok wskazanego komponentu EDIT na zakładce Standard

- Wciśnij klawisz **SHIFT**;
- Trzymając wciśnięty klawisz **SHIFT**, kliknij na ten komponent lewym klawiszem myszy (oznaczony komponent będzie wklęsły oraz otoczony błękitną obwolutą) – rysunek 5.6.2;



Rysunek 5.6.2. Widok wybranego komponentu EDIT

- Kliknij na formatce w tych miejscach, gdzie mają się znaleźć komponenty **EDIT**.

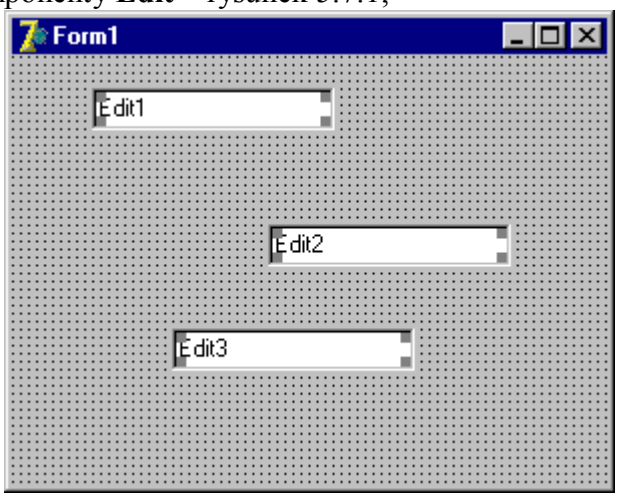
Po umieszczeniu wymaganej ilości tych samych komponentów, można umożliwiającą to funkcję wyłączyć. Wykonać to można klikając na inny komponent z palety komponentów lub klikając na klawisz znajdujący się po lewej stronie palety komponentów.

Ćwiczenie 5.7. Układanie komponentów umieszczonych na formatce

Umieść na formatce kilka tych samych komponentów, np. EDIT i wyrównaj je do lewej.

Sposób wykonania:

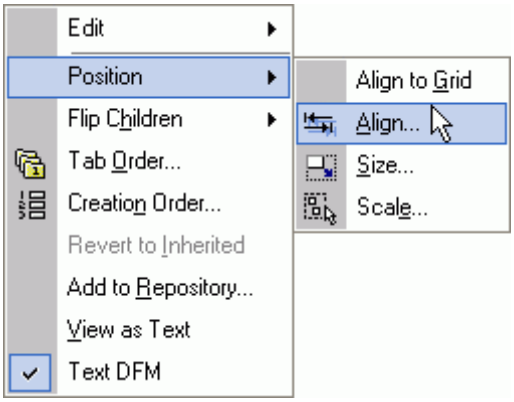
- Umieść na formie trzy komponenty **Edit** (zakładka **Standard**) - patrz ćwiczenie 5.6;
- Zaznacz trzy komponenty **Edit** – rysunek 5.7.1;



Rysunek 5.7.1. Widok zaznaczonych komponentów EDIT

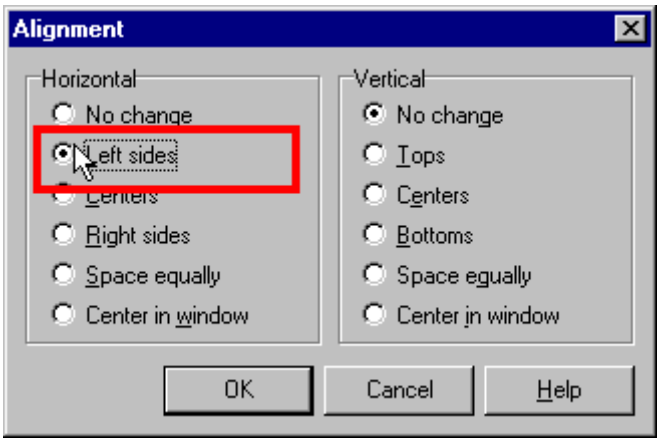
W celu zaznaczenia kilku komponentów, należy wcisnąć klawisz **SHIFT**. Następnie trzymając wciśnięty klawisz **SHIFT** klikać na kolejne komponenty, które mają być zaznaczone.

- Po zaznaczeniu komponentów, przesuń kursor nad dowolny wcześniej zaznaczony komponent i kliknij prawym klawiszem myszy;
- Z menu podręcznego wybierz **Position/Align…** (Położenie/Rozmieść) – rysunek 5.7.2;



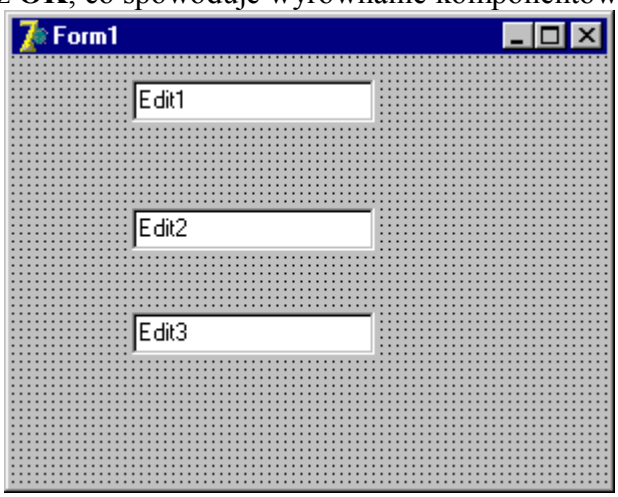
Rysunek 5.7.2. Widok wybranego polecenia Align (Rozmieść)

- W oknie **Alignment** (Rozmieszczenie) wybierz polecenie **Left sides** (Wyrównaj do lewej krawędzi), które znajduje się w kolumnie **Horizontal** (Poziomy) – rysunek 5.7.3;



Rysunek 5.7.3. Widok wybranej opcji Left sides (Wyrównaj do lewej krawędzi)

- Kliknij na klawisz **OK**, co spowoduje wyrównanie komponentów – rysunek 5.7.4;



Rysunek 5.7.4. Widok wyrównanych komponentów

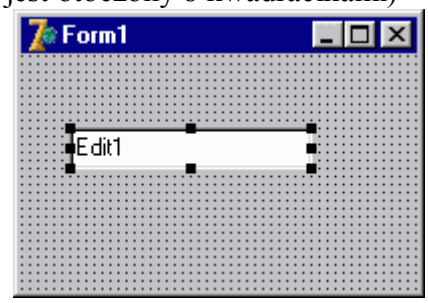
UWAGA: Wyrównywanie komponentów do lewej krawędzi, odbywa się względem lewej krawędzi komponentu, który został zaznaczony jako pierwszy.

Ćwiczenie 5.8. Zmiana koloru i nazwy komponentu

Umieść na formatce komponent, np. EDIT i zmień mu kolor na zielony oraz nadaj mu nazwę np. EditText.

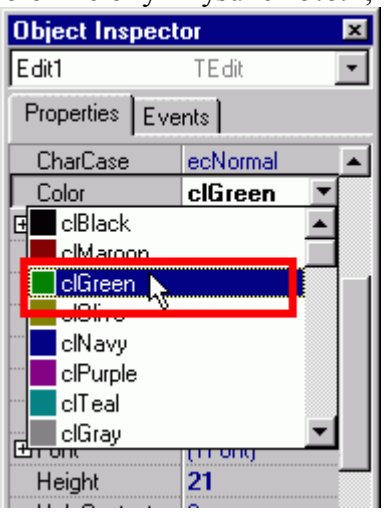
Sposób wykonania:

- Umieść komponent **EDIT** na formatce (patrz ćwiczenie 5.5);
- Zaznacz ten komponent, przez jednokrotne kliknięcie lewym klawiszem myszy na nim (zaznaczony komponent jest otoczony 8 kwadracikami) – rysunek 5.8.1;



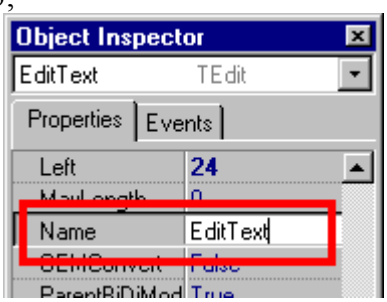
Rysunek 5.8.1. Widok zaznaczonego komponentu

- Naciśnij klawisz **F11**, co spowoduje przejście do okna **Object Inspector'a** (Inspektora obiektów) lub kliknij lewym klawiszem myszy na pasek tytułowy okna **Object Inspector'a** (Inspektora obiektów);
- Z listy właściwości (zakładka **Properties**) wybierz właściwość **Color** (Kolor);
- Rozwiń listę kolorów, klikając na klawisz (znajduje się z prawej strony napisu **Color** – Kolor) i wybierz kolor zielony – rysunek 5.8.2;



Rysunek 5.8.2. Widok rozwiniętej listy kolorów i wybranego kolor zielonego

- Wybierz właściwość **Name** (Nazwa) z listy właściwości i wprowadź nazwę „EditText" – rysunek 5.8.3;



Rysunek 5.8.3. Widok wprowadzonej nazwy

UWAGA: Właściwość komponentu **Name** (Nazwa) służy do nadania unikatowej nazwy komponentowi przez twórcę programu. Za pomocą tej nazwy programista może odwoływać się do właściwości lub zdarzeń tegoż komponentu.
W czasie umieszczania komponentów na formatce, DELPHI nadaje poszczególnym komponentom nazwy domyślne, np. wstawienie przycisku – nazwa domyślna „Button1", wstawienie listy – nazwa domyślna „ListBox1", itp.
Jeżeli na formatce znajduje się większa ilość, np. klawiszy, to DELPHI nadaje poszczególnym klawiszom nazwę „Button" i kolejny numer.
Na przykład: na formatce są umieszczone 4-ry przyciski, to DELPHI nadaje im domyślne następujące nazwy: „Button1", „Button2", „Button3", „Button4", itp.
Z innymi komponentami jest tak samo. Jak już wspomniałem wcześniej, domyślne nazwy komponentów nadane przez Delphi programista może zmienić we właściwości **Name** (Nazwa). Nazwa komponentu dla użytkownika jest niewidoczna, natomiast jest widoczny opis komponentu. W przypadku przycisku opis komponentu możemy umieścić dzięki właściwości **Caption** (Podpis). W innych komponentach, jak np. **Edit** opis umieszczany jest dzięki właściwości **Text** (Tekst).
Jak widać nazwa komponentu może być różna od opisu danego komponentu.

PAMIĘTAJ: W nazwie komponentu nie wolno używać polskich znaków!!!

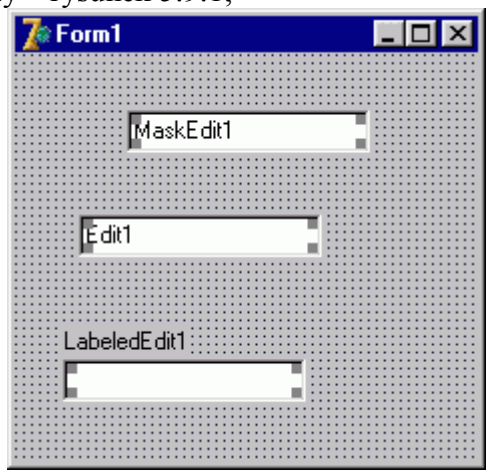
- Naciśnij klawisz ENTER, w celu zatwierdzenia nowej nazwy komponentu;
- Od tego momentu, do właściwości lub zdarzeń komponentu odwołuj się za pomocą nazwy „EditText”.

Ćwiczenie 5.9. Zmiana koloru grupie komponentów

Umieść na formatce następujące komponenty: Edit, MaskEdit oraz LabeledEdit i nadaj im kolor zielony.

Sposób wykonania:

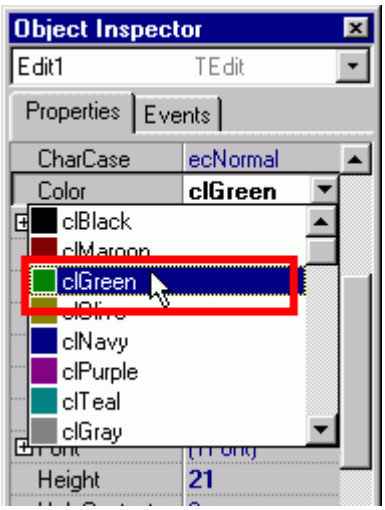
- Umieść komponenty **MaskEdit**, **Edit** oraz **LabeledEdit** na formatce (patrz ćwiczenie 5.5);
- Zaznacz te komponenty – rysunek 5.9.1;



Rysunek 5.9.1. Widok zaznaczonych komponentów EDIT

W celu zaznaczenia kilku komponentów, należy wcisnąć klawisz **SHIFT**. Następnie trzymając wciśnięty klawisz **SHIFT** klikać na kolejne komponenty, które mają być zaznaczone.

- Naciśnij klawisz **F11**, co spowoduje przejście do okna **Object Inspector’a** (Inspektora obiektów) lub kliknij lewym klawiszem myszy na pasek tytułowy okna **Object Inspector’a** (Inspektora obiektów);
- Z listy właściwości (zakładka **Properties** -Właściwości) wybierz pozycję **Color** (Kolor);
- Rozwiń listę kolorów, klikając na klawisz (znajduje się z prawej strony napisu **Color** – Kolor) i wybierz kolor zielony – rysunek 5.9.2;



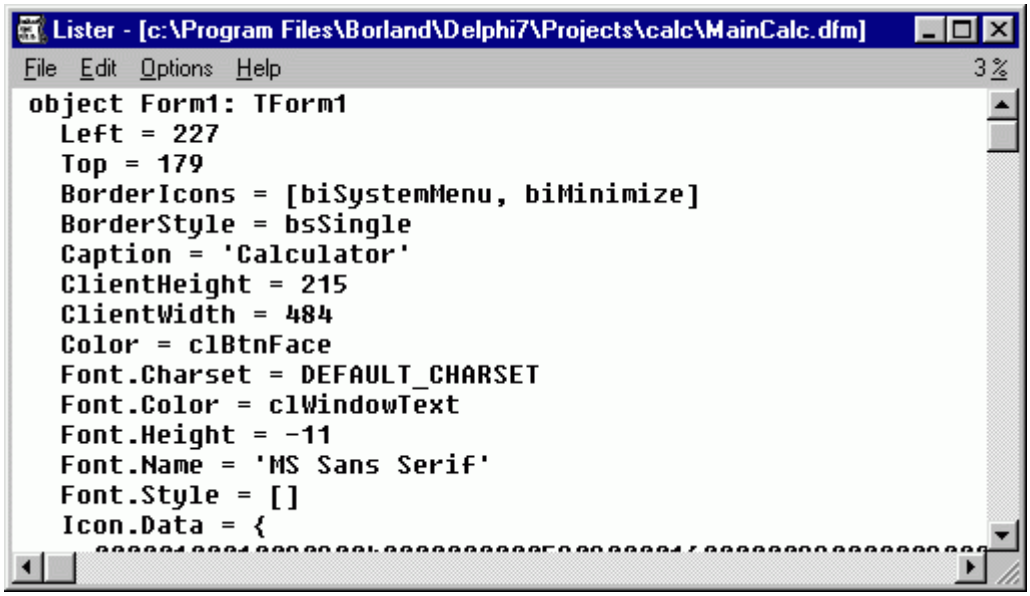
Rysunek 5.9.2. Widok rozwiniętej listy kolorów i wybranego kolor zielonego

UWAGA: Gdy są zaznaczone różne komponenty, to widoczne są tylko właściwości wspólne dla zaznaczonych komponentów.

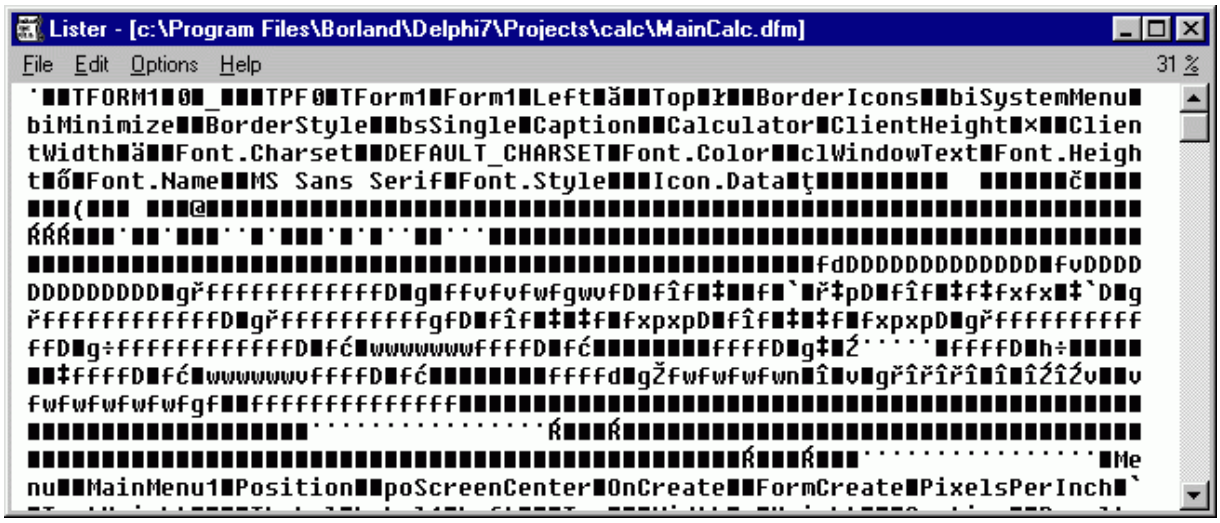
Ćwiczenie 5.10. Format zapisywania danych o formatce

Włącz binarny format zapisywania informacji o formatce.

Opis:
Informacje o komponentach umieszczanych na formatce są zapisywane w pliku o rozszerzeniu DFM. Plik ten od Delphi 4.0 zapisywany jest w formacie tekstowym – rysunek 5.10.1, tzn. można go edytować za pomocą dowolnego edytora tekstu. Natomiast w starszych wersjach Delphi, plik ten był zapisywany w formacie binarnym – rysunek 5.10.2, przez co mógł być on odczytywany tylko przez program Delphi.



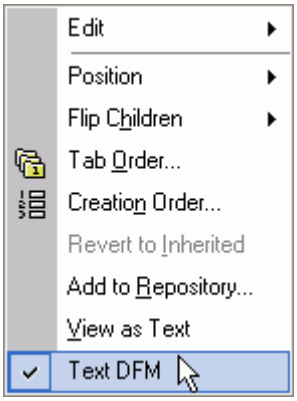
Rysunek 5.10.1. Widok pliku tekstowego z danymi na temat umieszczonych komponentów na formatce



Rysunek 5.10.2. Widok pliku binarnego z danymi na temat umieszczonych komponentów na formatce

Sposób wykonania:

- Kliknij prawym klawiszem myszy na formatce (domyślna nazwa **Form1**);
- Z menu podręcznego wybierz opcję **Text DFM** (Tekstowy plik DFM) – rysunek 5.10.3;



Rysunek 5.10.3. Widok wskazanej opcji Text DFM

- Kliknij na wybranej opcji prawym klawiszem myszy, co spowoduje jej odznaczenie;

Po tych czynności plik z informacjami o umieszczonych komponentach na formatce będzie zapisywany w pliku binarnym, również o rozszerzeniu DFM.

Ćwiczenie 5.11. Image - Wczytywanie rysunku

Zaprezentuj wczytywanie obrazka do komponentu IMAGE.

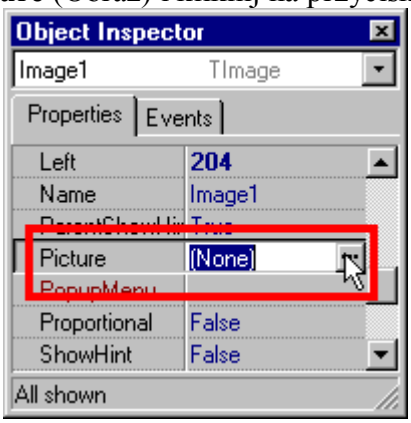
Opis:

Image jest komponentem, który służy do wyświetlania grafiki na ekranie. **Image** znajduje się na karcie **Additional** palety komponentów.

Sposób wykonania:

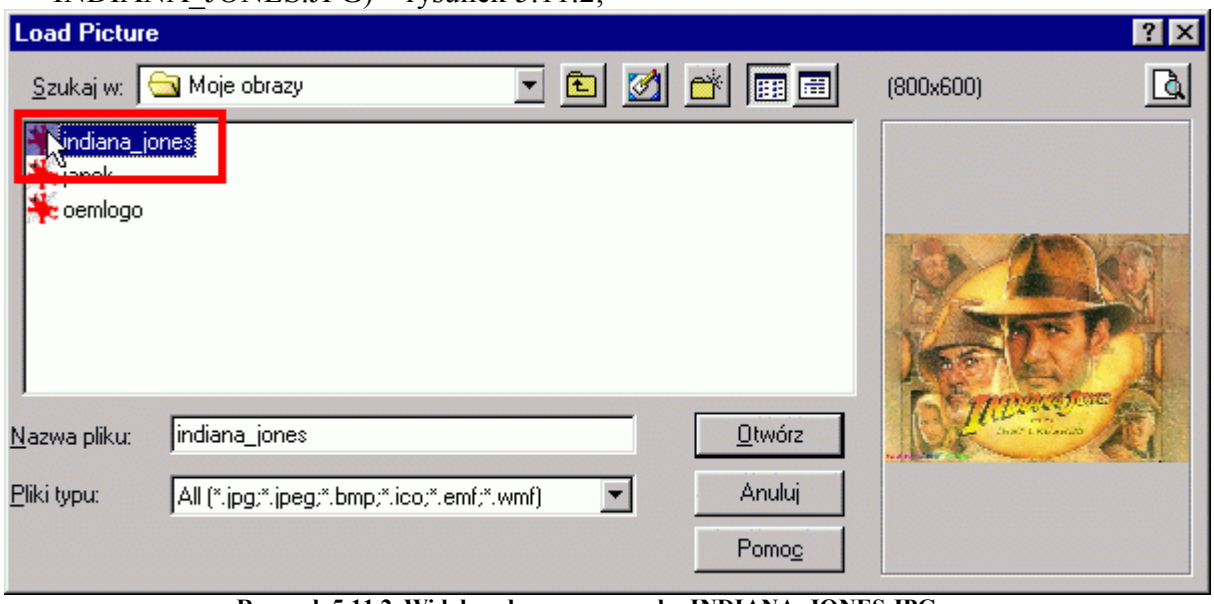
- Wybierz komponent **Image** z palety komponentów (karta **Additional**) i umieść go na formatce (patrz ćwiczenie 5.5);
- Przejdź do okna **Object Inspector** (Inspektor obiektów) naciskając klawisz funkcyjny **F11** lub wybierając menu **View/Object Inspector** (Widok/Inspektor obiektów);

- Będąc w oknie **Object Inspector** (Inspektor obiektów) wybierz zakładkę **Properties** (Właściwości);
- Wybierz właściwość **Picture** (Obraz) i kliknij na przycisk ... - rysunek 5.11.1;



Rysunek 5.11.1. Widok wskazanego klawisza

- W oknie **Picture editor** (Edytor rysunku) kliknij na klawisz **Load** (Wczytaj);
- W oknie **Load Picture** (Wczytaj rysunek) wybierz katalog, w którym znajdują się rysunki;
  Za pomocą listy rozwijanej **Szukaj w** możesz znaleźć katalog, w którym są rysunki (np. C:\MOJE DOKUMENTY\MOJE OBRAZY).
- Po odnalezieniu katalogu z rysunkami, wybierz dowolny rysunek (np. INDIANA_JONES.JPG) – rysunek 5.11.2;



Rysunek 5.11.2. Widok wybranego rysunku INDIANA_JONES.JPG

- Po wybraniu rysunku kliknij na klawisz **Otwórz**, co spowoduje wczytanie rysunku do okna **Picture editor** (Edytor rysunku);
- Będąc w oknie **Picture editor** (Edytor rysunku) kliknij na klawisz **OK**, co spowoduje wczytanie rysunku do komponentu **Image** (Rysunek).

Ćwiczenie 5.12. Button – generowanie zdarzenia naciśnięcia przycisku

Zaprezentuj działanie zdarzenia naciśnięcia przycisku.

Opis:

Button jest komponentem, który znajduje się na karcie **Standard** palety komponentów. Służy on, do uruchomienia jakiegoś zadanie w wyniku naciśnięcia tegoż przycisku.

Sposób wykonania:

- Wybierz komponent **Button** (karta **Standard**) i umieść go na formatce (patrz ćwiczenie 5.5);
- Kliknij dwukrotnie na nim (sposób generowania zdarzeń – ćwiczenie 5.2);
- W wygenerowanej procedurze wpisz linię „ShowMessage('To okienko ukazuje się po naciśnięciu klawisza');", poniżej zamieszczony jest przykład:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('To okienko ukazuje się po naciśnięciu klawisza');
end;
```

Ćwiczenie 5.13. Shape – proste figury geometryczne

Zaprezentuj możliwości rysowania figur geometrycznych komponentu SHAPE.

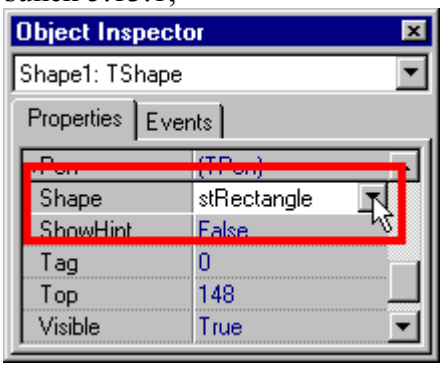
Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\SHAPE.

Opis:

Shape służy do wyświetlania prostych figur geometrycznych. Znajduje się on na karcie **Additional** palety komponentów.

Sposób wykonania:

- Wybierz komponent **Shape** (karta **Additional**) i umieść go na formatce (patrz ćwiczenie 5.5);
- Kliknij na komponent **Shape**, w celu zaznaczenia;
- Przejdź do okna **Object Inspector** (Inspektor obiektów) naciskając klawisz funkcyjny **F11** lub wybierając menu **View/Object Inspector** (Widok/Inspektor obiektów);
- Będąc w oknie **Object Inspector** (Inspektor obiektów) kliknij na zakładkę **Properties** (Właściwości);
- Wybierz właściwość **Shape** (Kształt);
- Kliknij na klawisz , aby rozwinąć listę kształtów (lista znajduje się z prawej strony właściwości **Shape**) – rysunek 5.13.1;



Rysunek 5.13.1. Widok wskazanego klawisza do rozwijania listy kształtów

- Po rozwinięciu listy możesz wybrać jeden z kilku dostępnych kształtów: Dostępne kształty:

stCircle (koło)	
stEllipse (Elipsa)	
StRectangle (Prostokąt)	
stRoundSquare (Kwadrat z zaokrąglonymi narożnikami)	
stSquare (Kwadrat)	
stRoundRect (Prostokąt z zaokrąglonymi narożnikami)	

- Po wybraniu dowolnego kształtu, komponent **Shape** (Kształt) automatycznie przybiera wygląd wybranego kształtu.

Przypisanie komponentowi odpowiedniego kształtu, jest również możliwe w kodzie programu. Wykonać to można wpisując np. „Shape1.Shape:= stRectangle;”, co spowoduje narysowanie prostokąta.
W celu określenia koloru konturu należy wpisać „Shape1.Pen.Color:= clGreen;”, natomiast kolor tła można określić wpisując „Shape1.Brush.Color:= clYellow;”.

Ćwiczenie 5.14. Panel

Zaprezentuj możliwości komponentu PANEL.

Opis:

Panel służy do tworzenia paska narzędzi, jak również umożliwia upiększenie tworzonego programu. Komponent ten znajduje się na karcie **Standard** palety komponentów.

Sposób wykonania:

- Wybierz komponent **Panel** (Karta **Standard**) i umieść go na formatce (patrz ćwiczenie 5.5);
- Kliknij na ten komponent, w celu zaznaczenia;
- Przejdź do okna **Object Inspector** (Inspektor obiektów) naciskając klawisz funkcyjny **F11** lub wybierając menu **View/Object Inspector** (Widok/Inspektor obiektów);
- Będąc w oknie **Object Inspector** (Inspektor obiektów) kliknij na zakładkę **Properties** (Właściwości);
- Za pomocą właściwości **BevelInner**, **BevelOuter** oraz **BorderStyle** można zmieniać wygląd komponentu **Panel**;

Wybrane wyglądy komponentu **Panel**, w zależności od ustawień właściwości:

BevelInner	BevelOuter	BorderStyle	Wygląd
bvLowered	bvRaised	BsNone	Panel1
bvRaised	bvLowered	BsNone	Panel1
bvNone	bvLowered	BsNone	Panel1
bvNone	bvLowered	BsNone	Panel1
bvRaised	bvRaised	BsSingle	Panel1

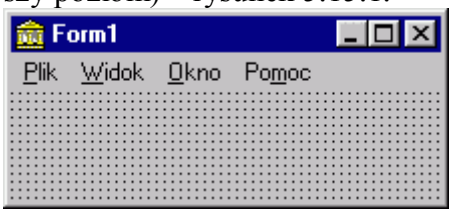
- Po wybraniu odpowiednich parametrów właściwości **BevelInner**, **BevelOuter** oraz **BorderStyle** komponent automatycznie przybiera wybrany wygląd.

Ćwiczenie 5.15. Menu

Zaprezentuj sposób dodania elementu do Menu.

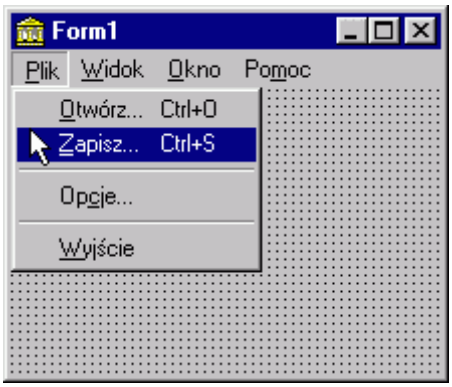
Opis:

Menu jest komponentem służącym do wyświetlania nazw funkcji programu oraz ich wywoływania. Menu składa się najczęściej z kilku elementów głównych ułożonych poziomo, tuż pod belką tytułową (pierwszy poziom) – rysunek 5.15.1.



Rysunek 5.15.1. Widok menu głównego z kilkoma elementami

Do elementów menu głównego podłączane są nowe elementy tworzące podmenu (drugi poziom) – rysunek 5.15.2.

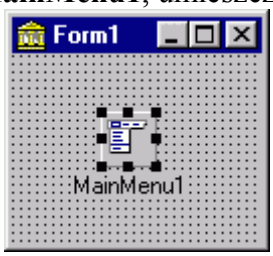


Rysunek 5.15.2. Widok podmenu, czyli elementów należących do elementu głównego „Plik”

Komponent **Menu** znajduje się na karcie **Standard** palety komponentów.

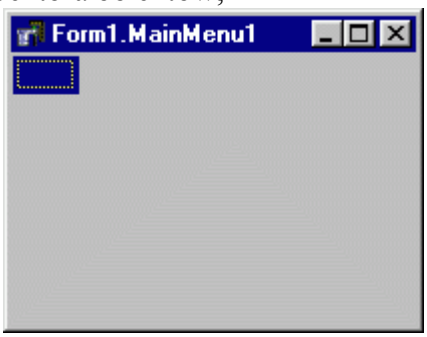
Tworzenie elementów menu głównego (pierwszy poziom):

- Wybierz komponent **MainMenu** z palety komponentów (karta **Standard**);
- Kliknij dwukrotnie na ikonę **MainMenu1**, umieszczoną na formie – rysunek 5.15.3;



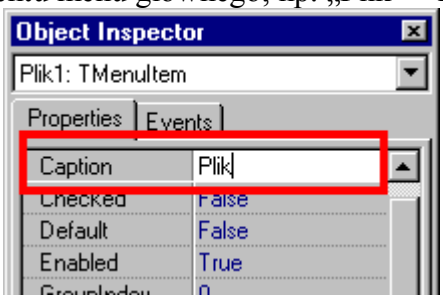
Rysunek 5.15.3. Widok ikony reprezentującej menu, która jest umieszczona na formatce

- Będąc w oknie **Form1.MainMenu1** – rysunek 5.15.4, przejdź do okna **Object Inspector** (Inspektora obiektów) naciskając klawisz funkcyjny **F11** lub kliknij na pasek tytułowy okna Inspektora obiektów;



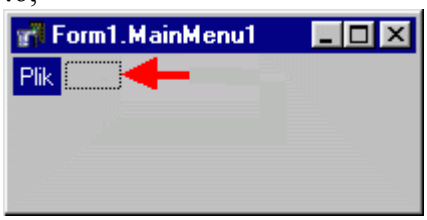
Rysunek 5.15.4. Widok okna umożliwiającego tworzenie menu z zaznaczonym pierwszym elementem menu głównego

- Będąc na zakładce **Properties** (Właściwości), wybierz właściwość **Caption** i wpisz nazwę pierwszego elementu menu głównego, np. „Plik" – rysunek 5.15.5;



Rysunek 5.15.5. Widok wpisanej nazwy „Plik" do właściwości Caption

- Po wpisaniu nazwy pierwszego elementu menu głównego, zostanie stworzony drugi pusty element menu głównego (pusty element oznaczony jest jako przerywana obwódka) – rysunek 5.15.6;



Rysunek 5.15.6. Widok drugiego pustego elementu menu głównego oznaczonego przerywaną obwódką

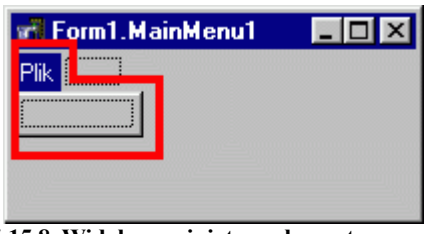
- Przejdź do drugiego elementu pustego, poprzez wybranie go klawiszami strzałkowymi lub kliknij na nim lewym klawiszem myszy (wybrany element będzie widoczny na niebieskim tle) – rysunek 5.15.7;

Rysunek 5.15.7. Widok zaznaczonego kolorem niebieskim pustego elementu

- Dalej postępuj tak samo, jak z elementem pierwszym (tzn. nadaj mu nazwę, itp.).

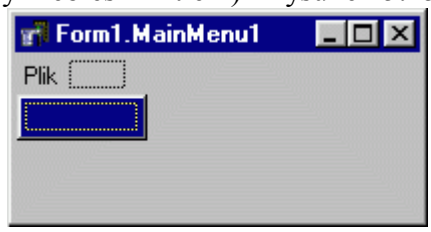
Podłączenie nowego elementu, pod element o nazwie „Plik" (drugi poziom):

- Rozwiń element o nazwie „Plik", postępując według następujących kroków:
  - Najedź na ten element za pomocą klawiszy strzałkowych lub kliknij na ten element lewym klawiszem myszy;
  - Po wybraniu elementu o nazwie „Plik" (będzie on oznaczony niebieskim tłem), zostanie wyświetlony element pusty (który należy do elementu o nazwie „Plik") - rysunek 5.15.8;



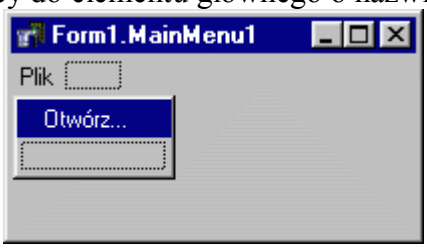
Rysunek 5.15.8. Widok rozwiniętego elementu o nazwie „Plik"

- Po ukazaniu się pustego elementu, wybierz go klawiszami strzałkowymi lub kliknij na nim (zostanie wyróżniony niebieskim tłem) – rysunek 5.15.9;



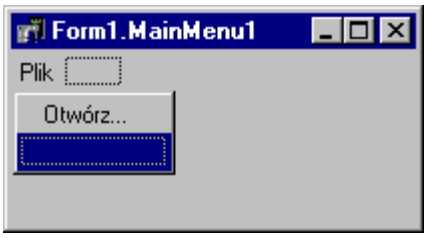
Rysunek 5.15.9. Widok zaznaczonego elementu pustego należącego do elementu o nazwie „Plik"

- Po wybraniu tego elementu, przejdź do okna **Object Inspector** (Inspektora obiektów) naciskając klawisz funkcyjny **F11** lub kliknij lewym klawiszem myszy na pasek tytułowy okna Inspektora obiektów;
- Będąc na zakładce **Properties** (Właściwości), wybierz właściwość **Caption** i wpisz nazwę pierwszego elementu, np. „Otwórz...";
- Po wpisaniu nazwy w pierwszym elemencie, zostanie wyświetlony drugi pusty element (również należący do elementu głównego o nazwie „Plik") – rysunek 5.15.10;



Rysunek 5.15.10. Widok nazwy pierwszego elementu i drugiego pustego elementu

- Przejdź do drugiego pustego elementu, za pomocą klawiszy strzałkowych lub kliknij na nim lewym klawiszem myszy;
- Po wybraniu drugiego elementu (zostanie on oznaczony kolorem niebieskim) – rysunek 5.15.11, postępuj tak samo jak z elementem pierwszym.



Rysunek 5.15.11. Widok zaznaczonego pustego elementu należącego do elementu o nazwie „Plik”

Jeżeli dana funkcja ma być wybierana za pomocą naciśnięcia jednego klawisza, to w nazwie funkcji należy wybrać dowolną literę i przed tą literą postawić znak „&”. Na przykład „Zapi&sz...”, co spowoduje, że litera ta będzie podkreślona „Zapisz...”.
Do wybranej funkcji można dodać klawisz skrótu za pomocą właściwości **ShortCut**. Skrót klawiszowy należy wybrać z listy dostępnych skrótów klawiszowych.

Przydatne informacje przy tworzeniu menu:

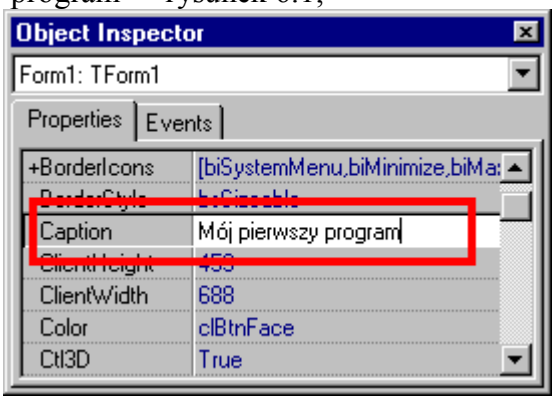
- Klawisz **Insert** – wstawia nową pozycję w menu między już istniejące;
- Klawisz **Delete** – usuwa wybraną pozycję z menu;
- Wstawienie znaku minus „-” we właściwości **Caption**, powoduje oddzielenie elementów za pomocą kreski.

6. Pierwszy program

Pierwszy program jaki napiszemy będzie wyświetlał słowa „Dzień dobry” oraz słowa „Do widzenia” w momencie kliknięcia na napis „Dzień dobry”.

W celu napisania pierwszego programu należy uruchomić Delphi, wybierając menu **START/PROGRAMY/BORLAND DELPHI 7/DELPHI 7** i wykonać następujące czynności:

- Kliknij na formatkę, w celu jej wybrania (jeżeli, jest na niej umieszczony i zaznaczony jakiś komponent);
- Naciśnij klawisz funkcyjny **F11**, aby przejść do okna **Object Inspector** (Inspektora obiektów);
- Będąc na zakładce **Properties** (Właściwości) wybierz właściwość **Caption** i wpisz tekst „Mój pierwszy program” – rysunek 6.1;

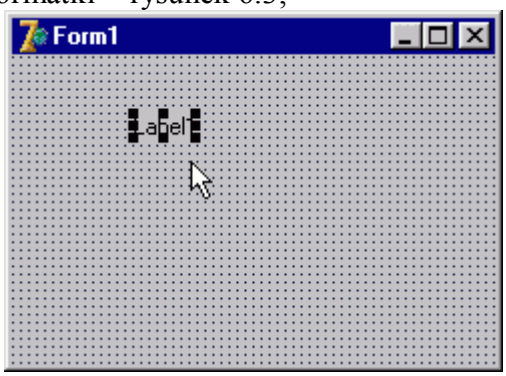


Rysunek 6.1. Widok wpisanego tekstu „Mój pierwszy program”

- Wybierz komponent **Label** (paleta komponentów **Standard**), przez kliknięcie na nim lewym klawiszem myszy - rysunek 6.2;

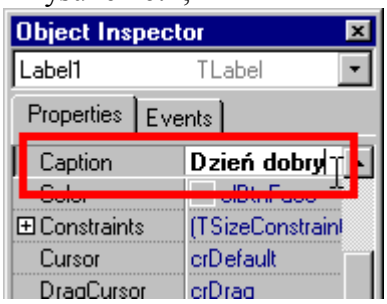
Rysunek 6.2. Widok wybranego komponentu Label

- Umieść komponent **Label** na formatce, przez kliknięcie lewym klawiszem myszy na dowolnym obszarze formatki - rysunek 6.3;



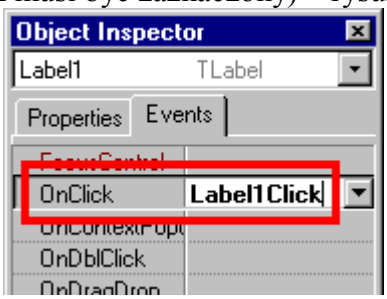
Rysunek 6.3. Widok umieszczonego komponentu Label na formatce

- Przejdź do okna **Object Inspector** (Inspektor Obiektów) naciskając klawisz funkcyjny **F11** i będąc na zakładce **Properties** (Właściwości), znajdź właściwość **Caption**;
- Kliknij lewym klawiszem myszy z prawej strony napisu **Caption** (druga kolumna) i wpisz tekst „Dzień dobry" - rysunek 6.4;



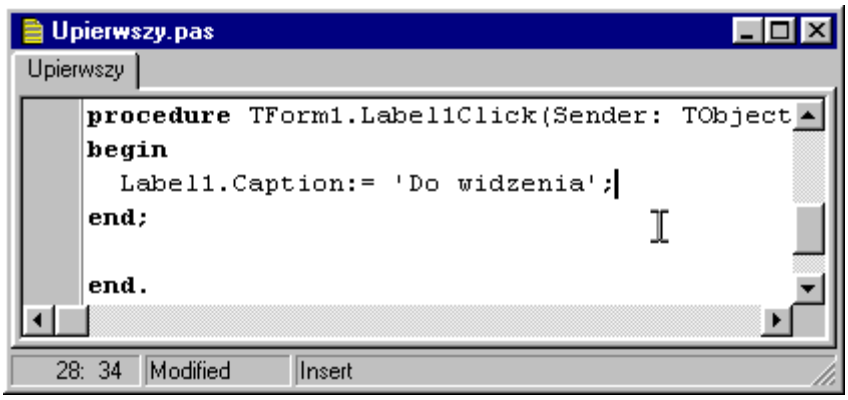
Rysunek 6.4. Widok wpisanego tekstu „Dzień dobry" z prawej strony napisu Caption

- Kliknij dwukrotnie lewym klawiszem myszy na komponencie **Label**, który znajduje się na formatce lub wybierz okno **Object Inspector** (Inspektora obiektów) i będąc na zakładce **Events** (Zdarzenia) kliknij z prawej strony nazwy zdarzenia **OnClick** (**uwaga**: komponent **Label** musi być zaznaczony) – rysunek 6.5;



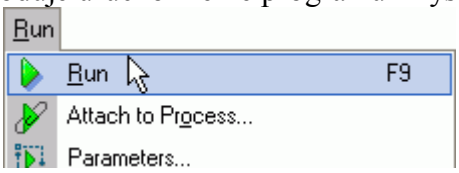
Rysunek 6.5. Widok wybranego zdarzenia OnClick

- W wygenerowanym zdarzeniu **OnClick** (w kodzie programu nosi nazwę **Label1Click**) wpisz linię „Label1.Caption:= 'Do widzenia';" – rysunek 6.6;

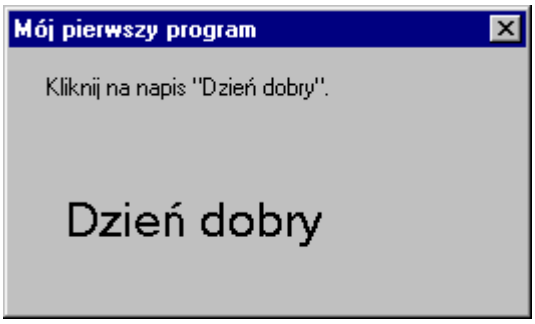


Rysunek 6.6. Widok wpisanej linii „Label1.Caption:= 'Do widzenia';”

- Naciśnij klawisz funkcyjny **F9** lub wybierz menu **Run/Run** (Uruchom/Uruchom) – rysunek 6.7., co spowoduje uruchomienie programu – rysunek 6.8.



Rysunek 6.7. Widok wybranego polecenia Run (Uruchom)



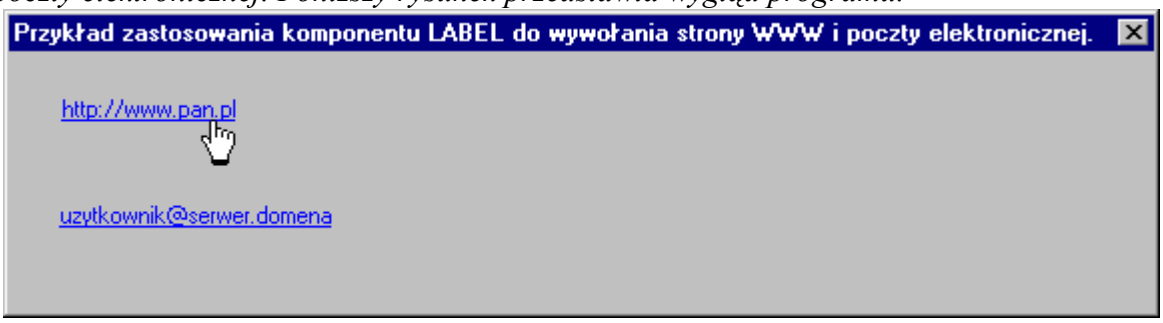
Rysunek 6.8. Widok działającego programu

Opisany przykład znajduje się w katalogu D7CW\PIERWSZY.

7. Ćwiczenia z podstawowych komponentów

Ćwiczenie 7.1. Label

Napisz program wykorzystując komponent Label do uruchomienia przeglądarki internetowej i poczty elektronicznej. Poniższy rysunek przedstawia wygląd programu.



Rysunek 7.1.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\LABEL.

Opis komponentu:

Label jest komponentem służącym do wyświetlania tekstu. Znajduje się on na karcie **Standard** palety komponentów.

Sposób wykonania:

- Wpisz nazwę biblioteki **ShellApi** w deklaracji **Uses**, np. **uses** ShellApi (bez tej deklaracji funkcja **ShellExecute** nie będzie działać);
- Wybierz dwa komponenty **Label** z palety komponentów (karta **Standard**) i umieść je na formatce jeden pod drugim;
- Kliknij dwukrotnie na formatce w celu wygenerowania zdarzenia **OnCreate**. Między słowami **begin** i **end** wpisz kod odpowiedzialny za ustawienie własności komponentów **Label**:

```
procedure TfrmForm1.FormCreate(Sender: TObject);
begin
  // Ustawienie własności komponentu Label1 i  Label2

  //-- Strona WWW --

  // Zamiana kursora myszki na rączkę w momencie najechania myszą
  // na komponent LABEL
  Label1.Cursor:= crHandPoint;

  Label1.Font.Color:= clBlue; // Ustawienie koloru czcionki na kolor niebieski.

  Label1.Font.Style:= [fsUnderline]; // Ustawienie podkreślenia pod napisem.

  Label1.Caption:= 'http://www.pan.pl'; // Wpisanie adresu strony WWW.
```

```
  //-- Poczta --

  // Zamiana kursora myszki na rączkę w momencie najechania
  // myszą na komponent LABEL
  Label2.Cursor:= crHandPoint;

  Label2.Font.Color:= clBlue; // Ustawienie koloru czcionek na kolor niebieski.

  Label2.Font.Style:= [fsUnderline]; // Ustawienie podkreślenia pod napisem.

  Label2.Caption:= 'uzytkownik@serwer.domena'; // Wpisanie adresu poczty.
end;
```

- Następnie kliknij dwukrotnie na komponent **Label1** w celu wygenerowania zdarzenia **OnClick** i wpisz kod odpowiedzialny za uruchomienie przeglądarki internetowej:

```
procedure TfrmForm1.Label1Click(Sender: TObject);
begin
  // Uruchomienie przeglądarki WWW.
  ShellExecute(GetDesktopWindow, 'open', PChar(Label1.Caption),
            nil, nil, SW_SHOWNORMAL);
end;
```

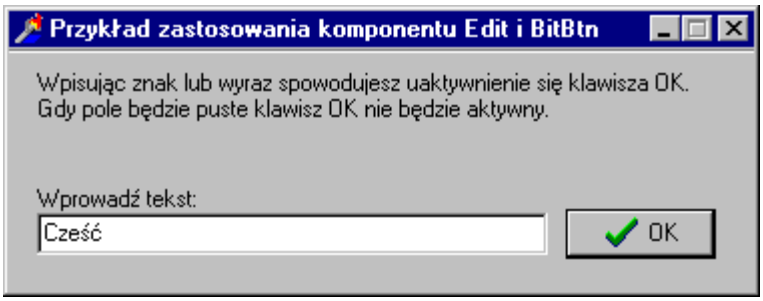
- Tak samo postępuj w przypadku komponentu **Label2** i wpisz kod odpowiedzialny za uruchomienie programu do obsługi poczty elektronicznej:

```
procedure TfrmForm1.Label2Click(Sender: TObject);
begin
  // Uruchomienie programu do obsługi poczty elektronicznej.
  ShellExecute(GetDesktopWindow, 'open', PChar('mailto:'+Label2.Caption),
            nil, nil, SW_SHOWNORMAL);
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.2. Edit, BitBtn

Napisz program, którego zadaniem jest uaktywnienie klawisza w momencie wprowadzenia tekstu. W innym przypadku klawisz ma być nieaktywny. Poniższy rysunek przedstawia wygląd takiego programu.



Rysunek 7.2.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\EDIT.

Opis komponentów:

Edit to komponent, który znajduje się na karcie **Standard** palety komponentów. Umożliwia on wprowadzanie tekstu w stworzonym programie.

BitBtn jest komponentem, który znajduje się na karcie **Additional** palety komponentów. Służy on do uruchomienia jakiegoś zadania w wyniku naciśnięcia tegoż przycisku. Komponent ten posiada właściwości komponentu **Button**.

Sposób wykonania:

- Wybierz komponent **Label** z palety komponentów (zakładka **Standard**);
- Mając zaznaczony komponent **Label**, w oknie Inspektora Obiektów we właściwości **Caption** wpisz „Wprowadź tekst:";
- Wybierz komponent **Edit** z palety komponentów (zakładka **Standard**);
- Wybierz komponent **BitBtn** (karta **Additional**) i umieść go na formatce obok **Edit'a** z prawej strony;
- Na formatce kliknij dwukrotnie w celu wygenerowania zdarzenia **OnCreate**. Następnie między słowami **begin** i **end** wpisz kod, który czyści zawartość Edit'a:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // FormCreate
  Edit1.Text:= ''; // Wyczyszczenie zawartości komponentu Edit

  Edit1Change(Sender); // Ponowne wykorzystanie zdarzenia Edit1Change
end;
```

- Kliknij dwukrotnie na komponent **Edit** znajdujący się na formatce, co spowoduje wygenerowanie zdarzenia **OnChange**. Między słowami **begin** i **end** wpisz kod kontrolujący aktywność klawisza **BitBtn**:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  // Aktywacja i dezaktywacja klawisza
  BitBtn1.Enabled:= FALSE; // Wyłączenie możliwości naciśnięcia klawisza.
  if (Length(Trim(Edit1.Text)) > 0) then
  begin
```

```
    {
     Włączenie możliwości naciśnięcia
     klawisza, jeżeli jest wpisany przynajmniej jeden znak.
    }
    BitBtn1.Enabled:= TRUE;
  end;
end;
```

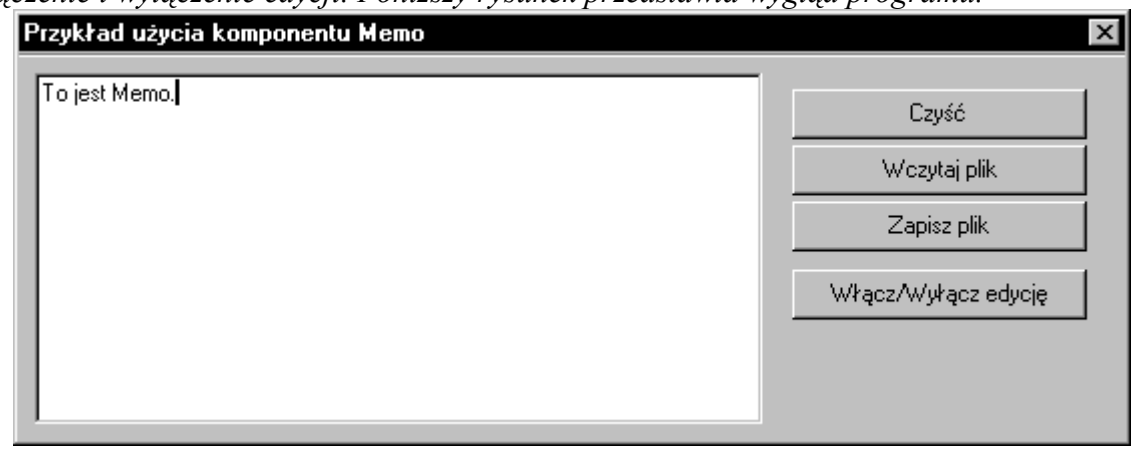
- Kliknij dwukrotnie na komponent **BitBtn** znajdujący się na formatce, co spowoduje wygenerowanie zdarzenia **OnClick**. W wygenerowanym zdarzeniu wpisz kod odpowiedzialny za wyświetlenie wprowadzonego tekstu do **Edit'a** na pasku tytułowym, po kliknięciu na komponent **BitBtn**:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  Caption:= Edit1.Text; // Wyświetlenie w pasku tytułowym zawartości komponentu Edit.
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.3. Memo

Napisz program, którego zadaniem będzie wczytanie i zapisanie pliku, wyczyszczenie oraz włączenie i wyłączenie edycji. Poniższy rysunek przedstawia wygląd programu.



Rysunek 7.3.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\MEMO.

Opis komponentu:
Memo jest polem edycyjnym służącym do edycji wielowierszowego tekstu bez formatowania. **Memo** można zastosować też do wyświetlania dużego tekstu, który nie mógłby być wyświetlany przy pomocy komponentu **Label**. **Memo** znajduje się na karcie **Standard** palety komponentów.

Sposób wykonania:

- Wybierz komponent **Memo** z palety komponentów (karta **Standard**);
- Wybierz komponent **Edit** (karta **Standard**);
- Wybierz kilka komponentów **Button** (karta **Standard**);

Nazwy poszczególnych przycisków są następujące:

- bCzysc – napis widoczny na przycisku to „Czyść”;
- bWczytajPlik – napis widoczny na przycisku to „Wczytaj plik”;
- bZapiszPlik – napis widoczny na przycisku to „Zapisz plik”;
- bWlaczWylaczEdycje - napis widoczny na przycisku to „Włącz/Wyłącz edycję”.

- Wybierz przycisk z napisem „Czyść” i kliknij na nim dwa razy (szybko) oraz wpisz kod do wygenerowanej procedury:

```
procedure TForm1.bCzyscClick(Sender: TObject);
begin
  Memo1.Lines.Clear; // Wyczyszczenie zawartości komponentu Memo
end;
```

- Kliknij dwukrotnie na przycisk z napisem „Wczytaj plik” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bWczytajPlikClick(Sender: TObject);
begin
  Memo1.Lines.Clear; // Wyczyszczenie zawartości komponentu Memo.

  if (FileExists('Memo.txt') = TRUE) then // Jeżeli plik istnieje to odczytaj go.
  begin
    Memo1.Lines.LoadFromFile('Memo.txt'); // Odczytanie pliku tekstowego "Memo.txt".
  end;
end;
```

- Kliknij dwukrotnie na przycisk z napisem „Zapisz plik” i wpisz kod do wygenerowanej procedury:

```
procedure TForm1.bZapiszPlikClick(Sender: TObject);
begin
  Memo1.Lines.SaveToFile('Memo.txt'); // Zapisanie do pliku tekstowego.
end;
```

- Kliknij dwukrotnie na przycisk z napisem „Włącz/Wyłącz edycję” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bWlaczWylaczEdycjeClick(Sender: TObject);
begin
  // Włącz/Wyłącz edycję
  if (Memo1.ReadOnly = FALSE) then
  begin
    Memo1.ReadOnly:= TRUE; // Wyłącza możliwość edytowania.
    Memo1.TabStop:= FALSE; // Umożliwia przechodzenie za pomocą klawisza TAB.
    Memo1.Color:= clBtnFace; // Ustawia na kolor szary.
  end
  else
  begin
    Memo1.ReadOnly:= FALSE; // Włącza możliwość edytowania.
```

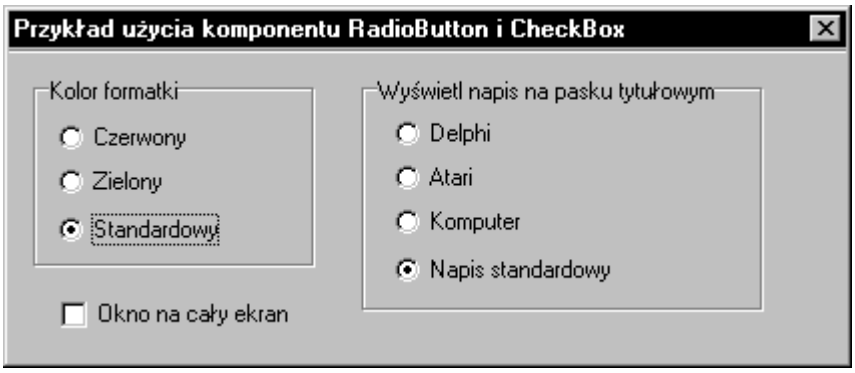
```
    Memo1.TabStop:= TRUE; // Włącza możliwość uaktywnienia za pomocą klawisza TAB.
    Memo1.Color:= clWindow; // Ustawia na kolor biały.
  end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Obsługa komponentu **RichEdit** jest podobna do komponentu **Memo**.

Ćwiczenie 7.4. RadioButton, GroupBox i CheckBox

Napisz program, który będzie zmieniał kolor formatki, napis na pasku tytułowym oraz powinien mieć możliwość maksymalizacji całego okna. Poniższy rysunek przedstawia wygląd programu.



Rysunek 7.4.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\RADIOBUT.

Opis komponentów:

RadioButton (przełącznik) jest komponentem służącym do wyboru jednej opcji z kilku dostępnych. Przełączniki te są reprezentowane przez białe kółeczka, które można grupować za pomocą komponentu GroupBox. Gdy przełącznik jest włączony, to białe kółeczko jest zapełnione mniejszym czarnym kółeczkiem, w innym przypadku jest nie zapełnione.

GroupBox jest komponentem służącym do grupowania np. komponentów **RadioButton**.

CheckBox (włącznik) jest komponentem służącym do wyboru jednej opcji niezależnie od innych opcji. Włącznik jest reprezentowany przez biały kwadracik. Gdy włącznik jest włączony, to w białym kwadraciku ukazuje się ptaszek, w innym przypadku kwadracik jest pusty. Włącznik może przyjmować trzy stany:

- włączony – wartość TRUE (ptaszek w środku białego kwadracika);
- wyłączony – wartość FALSE (kwadracik pusty);
- neutralny (kwadracik jest koloru szarego z ptaszkiem).

Standardowo włącznik może przyjmować dwa stany. Właściwość **AllowGrayed** jest ustawiona na wartość FALSE (domyślnie). W celu umożliwienia przyjmowania trzech stanów włącznika należy właściwość **AllowGrayed** ustawić na wartość TRUE.

Wyżej opisane komponenty znajdują się na zakładce **Standard** palety komponentów.

Sposób wykonania:

- Wybierz dwa komponenty **GroupBox** z palety komponentów (karta **Standard**);
- Wybierz kilka komponentów **RadioButton** (karta **Standard**);
- Wybierz **CheckBox** (karta **Standard**);
- Kliknij dwukrotnie na pierwszym **RadioButton'onie**, który znajduje się w pierwszej ramce. W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  // Daje kolor czerwony
  Form1.Color:= clRed;
end;
```

- Kliknij dwukrotnie na pierwszym **RadioButton'onie**, który znajduje się w drugiej ramce. W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.RadioButton4Click(Sender: TObject);
begin
  // Obsługa RadioButton'ów
  if (RadioButton4.Checked = TRUE) then Caption:= 'Delphi'; // Wyświetlenie napisu
  if (RadioButton5.Checked = TRUE) then Caption:= 'Atari';
  if (RadioButton6.Checked = TRUE) then Caption:= 'Komputer';
  if (RadioButton7.Checked = TRUE) then Caption:= 'Komponent RadioButton';
end;
```

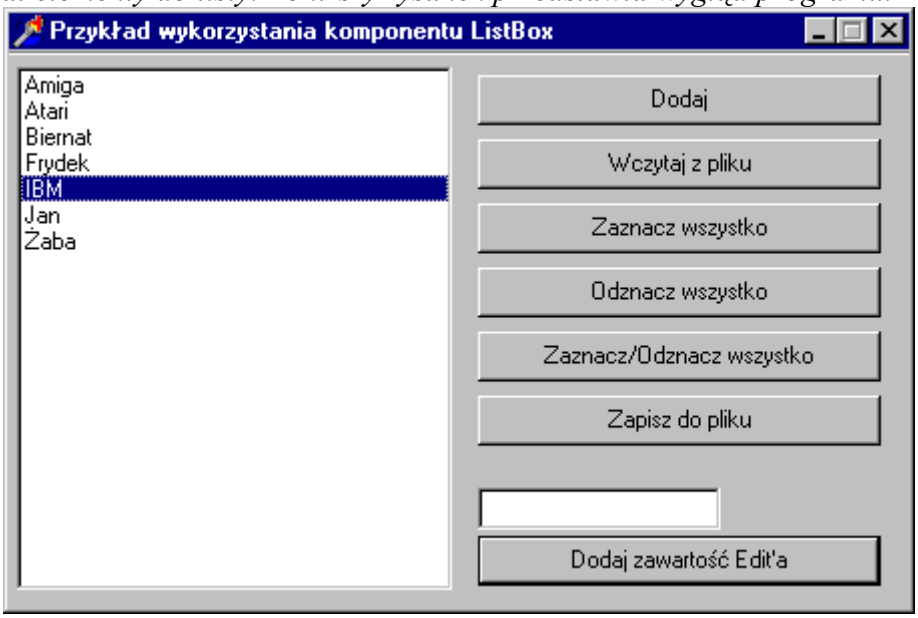
- Zaznacz drugi **RadioButton** (ramka 2) i mając go zaznaczony, w oknie **Object Inspector'a** (Inspektora Obiektów) wybierz zakładkę **Events** (Zdarzenia). Na zakładce tej wybierz zdarzenie **OnClick** i w liście rozwijanej wybierz procedurę **RadioButton4Click**. Tak postępuj z następnymi **RadioButton'ami** w ramce 2;
- Kliknij dwukrotnie na komponencie **CheckBox** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin

  if (CheckBox1.Checked = FALSE) then
  begin
    Form1.WindowState:= wsNormal; // Przywraca poprzedni stan okna.
  end
  else
  begin
    Form1.WindowState:= wsMaximized; // Daje okno na cały ekran.
  end;

end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.5. ListBox

Napisz program, który będzie wczytywał i zapisywał listę, zaznaczał i odznaczał elementy listy oraz dodawał elementy do listy. Poniższy rysunek przedstawia wygląd programu.



Rysunek 7.5.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\LBOX1.

Opis komponentu:

ListBox jest komponentem służącym do wyświetlania listy elementów. Elementy listy mogą być posortowane. Niestety elementów listy nie można edytować. Wybór większej ilości elementów dokonuje się trzymając klawisz **SHIFT** lub **CTRL**, co umożliwia zaznaczenie kilku elementów oddzielonych od siebie. Znajduje się on na karcie **Standard** palety komponentów.

Sposób wykonania:

- Wybierz komponent **ListBox** z palety komponentów (karta **Standard**);
- Wybierz komponent **Edit** (karta **Standard**);
- Wybierz kilka przycisków **Button** (karta **Standard**);
  Nazwy poszczególnych przycisków są następujące:
  - bDodaj – napis widoczny na przycisku to „Dodaj”;
  - bWczytajPlik – napis widoczny na przycisku to „Wczytaj z pliku”;
  - bZaznaczWszystko – napis widoczny na przycisku to „Zaznacz wszystko”;
  - bOdznaczWszystko – napis widoczny na przycisku to „Odznacz wszystko”;
  - bZaznaczOdznaczWszystko – napis widoczny na przycisku to „Zaznacz/Odznacz wszystko”;
  - bZapiszDoPliku – napis widoczny na przycisku to „Zapisz do pliku”;
  - bDodajZawartoscEdita – napis widoczny na przycisku to „Dodaj zawartość Edit’a”.

- Kliknij dwukrotnie na formatce i w wygenerowanym zdarzeniu **OnCreate** wpisz kod programu, który czyści zawartość listy i komponent **Edit**:

```
procedure TfrmMain.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Clear; // Wyczyszczenie zawartości komponentu ListBox

  Edit1.Text:= ''; // Czyści zawartość komponentu Edit.
end;
```

- Wybierz klawisz z napisem „Dodaj” i kliknij na nim dwukrotnie w celu wygenerowania zdarzenia **OnClick**. W wygenerowanym zdarzeniu wpisz kod odpowiedzialny za dodanie elementu do listy:

```
procedure TfrmMain.bDodajClick(Sender: TObject);
begin
  // Dodaj
  ListBox1.Items.Clear; // Czyści zawartość komponentu ListBox

  ListBox1.Items.Add('Spectrum'); // Dodaje pozycję do komponentu ListBox

  ListBox1.Items.Add('Amiga');
  ListBox1.Items.Add('IBM');
  ListBox1.Items.Add('Atari');
  ListBox1.Items.Add('Żyrafa');
  ListBox1.Items.Add('Cry');
  ListBox1.Items.Add('CPU');
  ListBox1.Items.Add('Komputer');

  ListBox1.Sorted:= TRUE; // Sortuje pozycje w komponencie ListBox
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Wczytaj z pliku” i w zdarzeniu **OnClick** wpisz kod odpowiedzialny za wczytanie listy z pliku:

```
procedure TfrmMain.bWczytajPlikClick(Sender: TObject);
begin
  // Wczytaj z pliku
  ListBox1.Items.Clear; // Czyści zawartość komponentu ListBox

  if (FileExists(Trim('Lista.txt'))=TRUE) then
  begin
    ListBox1.Items.LoadFromFile(Trim('Lista.txt')); // Wczytuje listę z pliku.

    ListBox1.Sorted:= TRUE; // Sortuje pozycje w komponencie ListBox
  end;
  {
    if (FileExists(Trim('Lista.txt'))=TRUE) then
    begin
    end;
```

```
  Funkcja ta sprawdza, czy na dysku istnieje plik o
  nazwie "Lista.txt", jeżeli istnieje, to go  wczytuje, w
  przeciwnym przypadku nie wykonuje nic.
  }
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Zaznacz/Odznacz wszystko” i w wygenerowanym zdarzeniu wpisz kod:

```
procedure TfrmMain.bZaznaczOdznaczWszystkoClick(Sender: TObject);
var
  TT: Integer;
begin
  // Zaznacza/Odznacza wszystko

   // Włącza możliwość zaznaczania więcej niż jednej pozycji.
  ListBox1.MultiSelect:= TRUE;

  for TT:= 0 to ListBox1.Items.Count-1 do
    if (ListBox1.Selected[TT] = TRUE) then
    begin
      ListBox1.Selected[TT]:= FALSE;
      {
       ListBox1.Selected[TT]:= FALSE;
       Odznacza pozycję o podanym numerze, który znajduje się pod zmienną "TT",
      jeżeli był zaznaczony.
      }
    end
    else
    begin
      ListBox1.Selected[TT]:= TRUE;
      {
       ListBox1.Selected[TT]:= FALSE;
       Zaznacza pozycję o podanym numerze, który znajduje się pod zmienną "TT",
      jeżeli był odznaczony.
      }
    end;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Zapisz do pliku” i w wygenerowanym zdarzeniu wpisz kod:

```
procedure TfrmMain.bZapiszDoPlikuClick(Sender: TObject);
begin
  // Zapisz do pliku
  ListBox1.Items.SaveToFile('Lista2.txt');
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Zapisz do pliku” i w wygenerowanym zdarzeniu wpisz kod:

```
procedure TfrmMain.bDodajZawartoscEditaClick(Sender: TObject);
begin
  // Dodaj zawartość Edit'a do ListBox'u

  {
   W momencie spełnienia warunku, tzn. gdy komponent Edit będzie
   zawierał chociaż jeden znak, to nastąpi dodanie tego znaku do ListBox'u.
  }
  if (Trim(Edit1.Text)<>'') then
  begin
    ListBox1.Items.Add(Trim(Edit1.Text)); // Dodaje pozycję do komponentu ListBox.
    ListBox1.Sorted:= TRUE; // Sortuje pozycje w komponencie ListBox
    Edit1.Text:= ''; // Czyści zawartość komponentu Edit
  end;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Odznacz wszystko” i w wygenerowanym zdarzeniu wpisz kod:

```
procedure TfrmMain.bOdznaczWszystkoClick(Sender: TObject);
var
  TT: Integer;
begin
  // Odznacza wszystko

  // Włącza możliwość zaznaczania więcej niż jednej pozycji w komponencie ListBox
  ListBox1.MultiSelect:= TRUE;

// Odznaczenie wszystkich elementów
for TT:= 0 to ListBox1.Items.Count-1 do
    ListBox1.Selected[TT]:= FALSE;
  {
   ListBox1.Selected[TT]:= FALSE;
   Odznacza pozycję o podanym numerze, który znajduje się pod zmienną "TT".
  }
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Zaznacz wszystko” i w wygenerowanym zdarzeniu wpisz kod:

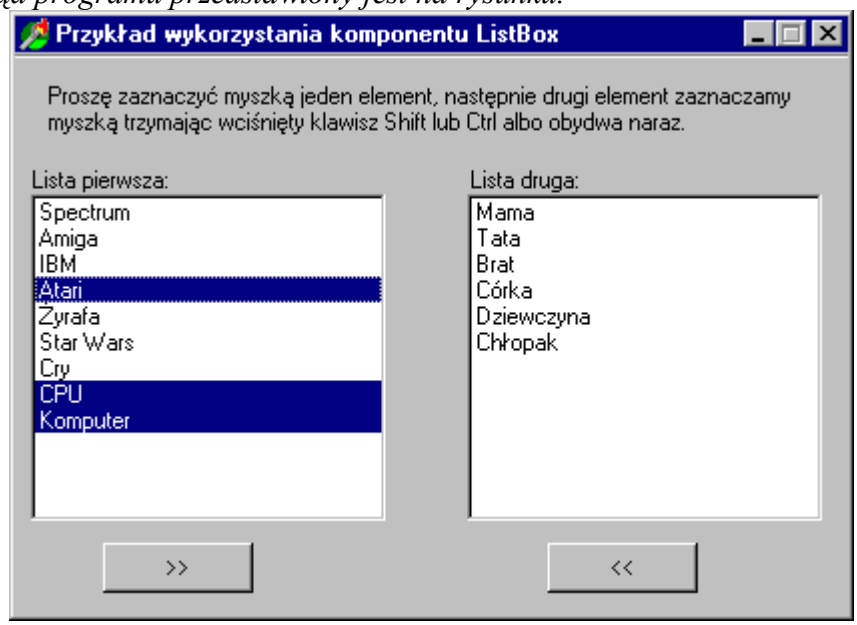
```
procedure TfrmMain.bZaznaczWszystkoClick(Sender: TObject);
var
  TT: Integer;
begin
  // Zaznacza wszystko

  // Włącza możliwość zaznaczania więcej niż jednej pozycji w komponencie ListBox1
```

```
  ListBox1.MultiSelect:= TRUE;

  // Zaznaczenie wszystkich elementów
  for TT:= 0 to ListBox1.Items.Count-1 do
    ListBox1.Selected[TT]:= TRUE;

  // ListBox1.Selected[4]:= TRUE; - Zaznacza pozycję o nr 4
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.6. ListBox – kopiowanie między listami

Napisać program, w którym będzie możliwa zamiana zaznaczonych elementów między dwoma listami. Wygląd programu przedstawiony jest na rysunku.



Rysunek 7.6.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\LBOX3.

Sposób wykonania:

- Wybierz dwa komponenty **ListBox** z palety komponentów (karta **Standard**) i umieść je obok siebie;
- Wybierz dwa komponenty **Button** (karta **Standard**);
  Nazwy poszczególnych przycisków są następujące:
  - bDoListy2 – napis widoczny na przycisku to „>>”;
  - bDoListy1 - napis widoczny na przycisku to „<<”.
- Kliknij dwukrotnie na formatce i w wygenerowanej procedurze wpisz kod:

```
procedure TfrmMain.FormCreate(Sender: TObject);
begin
  // Lista 1
  ListBox1.Items.Clear; // Czyści zawartość komponentu ListBox
```

```
  ListBox1.Items.Add('Spectrum'); // Dodaje pozycję do komponentu ListBox
  ListBox1.Items.Add('Amiga');
  ListBox1.Items.Add('IBM');
  ListBox1.Items.Add('Atari');
  ListBox1.Items.Add('Żyrafa');
  ListBox1.Items.Add('Star Wars');
  ListBox1.Items.Add('Cry');
  ListBox1.Items.Add('CPU');
  ListBox1.Items.Add('Komputer');

  // Lista 2
  ListBox2.Items.Clear; // Czyści zawartość komponentu ListBox

  ListBox2.Items.Add('Mama'); // Dodaje pozycję do komponentu ListBox
  ListBox2.Items.Add('Tata');
  ListBox2.Items.Add('Brat');
  ListBox2.Items.Add('Córka');
  ListBox2.Items.Add('Dziewczyna');
  ListBox2.Items.Add('Chłopak');

  {
   Włącza możliwość zaznaczania więcej
   niż jednej pozycji w komponencie ListBox
  }
  ListBox1.MultiSelect:= TRUE;
  ListBox2.MultiSelect:= TRUE;
end;
```

- Kliknij dwukrotnie na klawiszu „>>” i w wygenerowanej procedurze wpisz kod:

```
procedure TfrmMain.bDoListy2Click(Sender: TObject);
var
  TT: Integer;
begin
  // Przenosi elementy z listy 1 do listy 2
  for TT:= ListBox1.Items.Count-1 downto 0 do
    if (ListBox1.Selected[TT] =TRUE) then
    begin
      ListBox2.Items.Add(Trim(ListBox1.Items[TT]));
      ListBox1.Items.Delete(TT);
    end;
   {
     for TT:= ListBox1.Items.Count-1 downto 0 do

    Przy usuwaniu elementu z listy należy tę listę przeglądać od elementu o
    najwyższym numerze do elementu o najniższym numerze.
    Przeglądanie od elementu o najniższym numerze do elementu o najwyższym numerze
    spowoduje błąd, przez wyjście poza zakres tablicy w wyniku próby usunięcia elementu o
   numerze nie istniejącym.
   }
```

```
end;
```

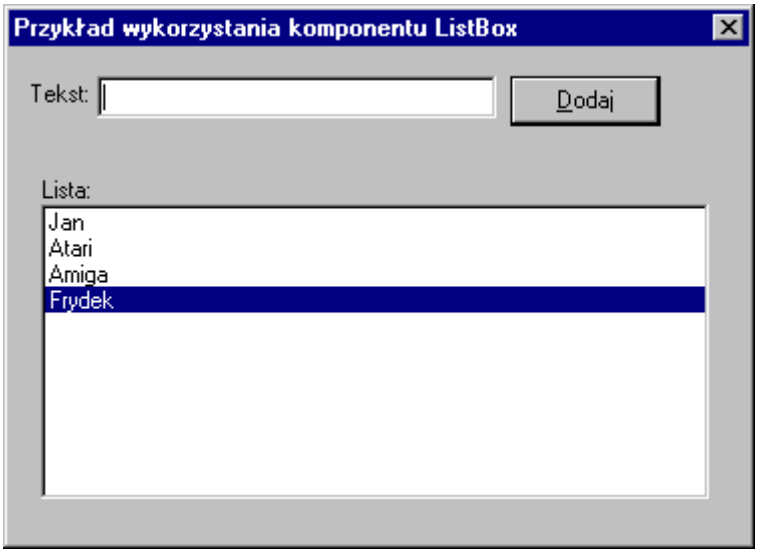
- Kliknij dwukrotnie na klawiszu „<<” i w wygenerowanej procedurze wpisz kod:

```
procedure TfrmMain.bDoList1Click(Sender: TObject);
var
  TT: Integer;
begin
  // Przenosi elementy z listy 2 do listy 1
  for TT:= ListBox2.Items.Count-1 downto 0 do
    if (ListBox2.Selected[TT] =TRUE) then
    begin
      ListBox1.Items.Add(Trim(ListBox2.Items[TT]));
      ListBox2.Items.Delete(TT);
    end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.7. ListBox – dodawanie do listy

Napisać program, który będzie zaznaczał dodany element do listy oraz nie będzie dopuszczał do dodania dwóch takich samych elementów. Poniższy rysunek przedstawia wygląd programu.



Rysunek 7.7.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\LBOX4.

Sposób wykonania:

- Wybierz komponent **Label** (karta **Standard**);
- Wybierz komponent **Edit** (karta **Standard**);
- Wybierz przycisk **Button** (karta **Standard**), nazwij go „bAdd” i opisz jako „Dodaj”;
- Wybierz komponent **ListBox** z palety komponentów (karta **Standard**);

- Napisz funkcję zapobiegającą dodaniu dwóch takich samych elementów:

```
function TForm1.ListBoxVerify(txtTextVerify: String): Boolean;
var
  TT :Integer;
begin
  {
   Sprawdza, czy w liście istnieje ciąg znaków wpisanych w komponencie Edit.
   Jeżeli wpisany ciąg znaków istnieje, to funkcja zwróci
   wartość TRUE (Prawda), w innym przypadku zwróci FALSE.

   AnsiUpperCase - zmienia cały ciąg znaków na duże litery bez względu na język.
   Trim - likwiduje spacje po obu stronach ciągu znaków.
  }
  ListBoxVerify:= FALSE;
  for TT:= ListBox1.Items.Count-1 downto 0 do
    if (AnsiUpperCase(Trim(txtTextVerify)) =
        AnsiUpperCase(Trim(ListBox1.Items[TT]))) then ListBoxVerify:= TRUE;
end;
```

- Napisz funkcję zaznaczającą element dodany do listy:

```
function TForm1.ListBoxSelectAdd(txtAddText: String): String;
var
  TT: Integer;
begin
  {
   Dodanie nowego ciągu znaków z zaznaczeniem dodawanego ciągu znaków.
   Jeżeli nastąpiło dodanie ciągu znaków to funkcja zwróci dodany ciąg
   znaków, w przeciwnym przypadku nastąpi zwrócenie znaku pustego.
  }
  ListBoxSelectAdd:= '';
  ListBox1.MultiSelect:= TRUE; // Ustawienie możliwości zaznaczania kilku pozycji.

  {
   Ustawienie wysokości, która umożliwia widoczność ostatniego elementu listy.
  }
  ListBox1.IntegralHeight:= TRUE;

  ListBox1.Sorted:= FALSE; // Wyłączenie sortowania.

  // Dodanie ciągu znaków, jeżeli zmienna "txtAddText" nie jest pusta.
  if (Trim(txtAddText)<>'') then
  begin
    ListBox1.Items.Add(Trim(txtAddText)); // Dodanie ciągu znaków.


    for TT:= 0 to ListBox1.Items.Count-1 do
      ListBox1.Selected[TT]:= FALSE; // Odznaczenie wszystkich pozycji w liście.
```

```
    // Zaznaczenie dodanej pozycji (ciągu znaków).
    ListBox1.Selected[ListBox1.Items.Count-1]:= TRUE;

    ListBoxSelectAdd:= Trim(txtAddText); // Zwrócenie dodanego ciągu znaków.

  end;
end;
```

♦ Zadeklaruj wcześniej napisane funkcje w sekcji **private** (prywatny):

```
type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    function ListBoxVerify(txtTextVerify: String): Boolean;
    function ListBoxSelectAdd(txtAddText: String): String;
  public
    { Public declarations }
  end;
```

♦ Kliknij dwukrotnie na formatkę i w wygenerowanym zdarzeniu wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin

  Edit1.Text:= ''; // Wyczyszczenie komponentu Edit.
  ListBox1.Items.Clear; // Wyczyszczenie listy.
end;
```

♦ Kliknij dwukrotnie na komponent **Edit** i w wygenerowanym zdarzeniu wpisz kod:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  {
   Edit1Change - w tej procedurze wpisujemy rozkazy, które
   będą wykonywane w momencie zmiany zawartości komponentu Edit.
  }

  bAdd.Enabled:= FALSE; // Wyłączenie przycisku DODAJ.

  {
    if...then
    Sprawdza, czy jest wpisany chociaż jeden znak.
    Jeżeli tak to zostanie uaktywniony przycisk DODAJ.
  }
  if (Length(Trim(Edit1.Text)) > 0) then
    bAdd.Enabled:= TRUE; // Włączenie przycisku DODAJ.
end;
```

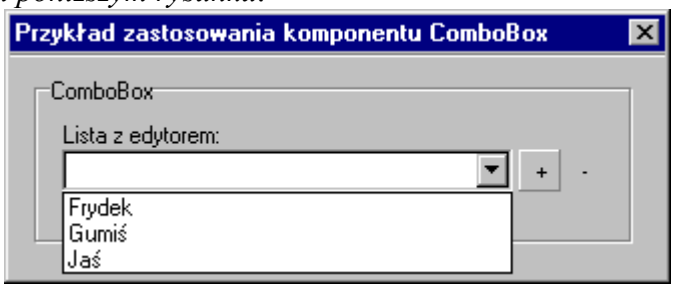
- Kliknij dwukrotnie na klawisz z napisem „Dodaj” i w wygenerowanym zdarzeniu wpisz kod:

```
procedure TForm1.bAddClick(Sender: TObject);
begin
  // Dodanie ciągu znaków do listy.

  if (ListBoxVerify(Edit1.Text) = FALSE) and
     (Trim(Edit1.Text)<>'') then
  begin
    {
     Dodanie nowego ciągu znaków nastąpi w momencie, gdy
     tego znaku nie ma w liście i zawartość komponentu Edit
     nie jest pusta. W przeciwnym przypadku nie nastąpi
     dodanie do listy ciągu znaków.
    }
    ListBoxSelectAdd(Edit1.Text);
    Edit1.Text:= ''; // Wyczyszczenie komponentu Edit.
  end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.8. ComboBox

Stworzyć listę za pomocą komponentu ComboBox z możliwością dodania (bez możliwości dodania dwóch takich samych elementów) i usunięcia elementu z listy. Wygląd programu przedstawiony jest na poniższym rysunku.



Rysunek 7.8.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\COMBOBOX.

Opis komponentu:

ComboBox znajduje się na karcie **Standard** palety komponentów. Jest to lista rozwijana połączona z polem edycyjnym. Rozwinięcie listy następuje po kliknięciu na strzałkę skierowaną w dół (znajduje się ona z prawej strony pola edycyjnego) lub naciśnięciu kombinacji klawiszy **Alt+Strzałka w dół**. Wybranie elementu z listy, powoduje umieszczenie tego elementu w polu edycyjnym przy jednoczesnym zamknięciu listy.

Sposób wykonania:

- Wybierz komponent **Label** z palety komponentów (karta **Standard**);
- Zaznacz komponent **Label** klikając na nim lewym klawiszem myszy i naciśnij klawisz funkcyjny **F11**, aby przejść do okna **Object Inspector** (Inspektora Obiektów);
- Będąc na zakładce **Properties** (Właściwości) wybierz właściwość **Caption** komponentu **Label** i wpisz „Lista z edytorem:";
- Wybierz komponent **ComboBox** z palety komponentów (karta **Standard**);
- Wybierz komponent **SpeedButton** z palety komponentów (karta **Additional**) i wykonaj następujące kroki:
  - Umieść go z prawej strony komponentu **ComboBox**;
  - Naciśnij klawisz funkcyjny **F11**, aby przejść do okna **Object Inspector** (Inspektora obiektów);
  - Będąc na zakładce **Properties** (Właściwości) wybierz właściwość **Name** i wpisz tam nazwę przycisku „sbDodaj";
  - Przejdź do właściwości **Caption** i wpisz opis przycisku „+";
  - Przejdź do właściwości **Hint** i wpisz „Dodaj do listy";
  - Na koniec przejdź do właściwości **ShowHint** i nadaj jej wartość TRUE (TRUE – pokazuje podpowiedzi).
- Wybierz drugi komponent **SpeedButton** i wykonaj następujące kroki:
  - Umieść go tuż za pierwszym klawiszem;
  - Naciśnij klawisz funkcyjny **F11**, aby przejść do okna **Object Inspector** (Inspektora obiektów);
  - Będąc na zakładce **Properties** (Właściwości) wybierz właściwość **Name** i wpisz tam nazwę przycisku „sbUsun";
  - Przejdź do właściwości **Caption** i wpisz opis przycisku „-";
  - Przejdź do właściwości **Hint** i wpisz „Usuń z listy";
  - Na koniec przejdź do właściwości **ShowHint** i nadaj jej wartość TRUE (TRUE – pokazuje podpowiedzi).
- Kliknij dwukrotnie na formatce, co spowoduje wygenerowanie zdarzenia **OnCreate** i wpisz w nim następujący fragment kodu:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Items.Clear; // Czyszczenie zawartości listy.
  ComboBox1.Text:= ''; // Wyczyszczenie zawartości edytora.

  // Wczytanie listy z pliku.
  if (FileExists('Lista.txt') = TRUE) then  // Sprawdzenie czy istnieje plik na dysku.
    ComboBox1.Items.LoadFromFile('Lista.txt'); // Odczytanie listy z pliku.
end;
```

- Następnie napisz funkcję, która sprawdzi, czy nie próbujesz dodać dwóch takich samych elementów:

```
function TForm1.ComboBoxVerify: Boolean;
var
  TT :Integer; // Deklaracja zmiennej
begin
  {
   Funkcja sprawdza, czy wpisany napis w edytorze znajduje się w liście.
```

```
  Jeżeli wpisany napis jest w liście to funkcja zwróci wartość TRUE (Prawda),
  w innym przypadku funkcja zwróci FALSE (Fałsz).
 }
 ComboBoxVerify:= FALSE;
 for TT:= 0 to ComboBox1.Items.Count-1 do
  if (AnsiUpperCase(Trim(ComboBox1.Text)) =
     AnsiUpperCase(Trim(ComboBox1.Items[TT]))) then ComboBoxVerify:= TRUE;

 // AnsiUpperCase() – Zamiana znaków dowolnej wielkości na duże, bez
 // względu   na język.
 // Trim() – Likwiduje spacje po lewej i prawej stronie ciągu znaków.
end;
```

- Zadeklaruj stworzoną wcześniej funkcję w typie obiektowym:

```
type
 TForm1 = class(TForm)
  function ComboBoxVerify: Boolean; // Deklaracja funkcji sprawdzającej
  procedure FormCreate(Sender: TObject);
 private
end;
```

- Kliknij dwukrotnie na klawisz z napisem „+”, co spowoduje wygenerowanie procedury. Między słowami **begin** i **end** wpisz kod odpowiedzialny za dodanie elementu do listy:

```
procedure TForm1.sbDodajClick(Sender: TObject);
begin
 // Dodaj do listy
 if (ComboBoxVerify = FALSE) and (Trim(ComboBox1.Text)<>'') then
 begin
  {
   Dodanie do listy nastąpi w momencie, gdy edytor listy nie
   będzie pusty i napisany ciąg znaków nie będzie występował w liście.
   Jest to warunek, który jest sprawdzany za pomocą instrukcji If...Then.
  }
  ComboBox1.Items.Add(ComboBox1.Text); // Dodanie nowego elementu do listy.

  ComboBox1.Sorted:= TRUE; // Włączenie sortowania listy.

  ComboBox1.Items.SaveToFile('Lista.txt'); // Zapisanie listy do pliku.
 end;

 ComboBox1.SelectAll; // Zaznaczenie ciągu znaków w edytorze.

 ComboBox1.SetFocus; // Ustawienie listy z edytorem jako aktywny.
end;
```

- Kliknij na komponent **ComboBox**, w celu zaznaczenia go i wykonaj następujące kroki:

- Naciśnij klawisz funkcyjny **F11**, w celu przejścia do okna **Object Inspector** (Inspektora obiektów);
- Będąc na zakładce **Events** (Zdarzenia) kliknij dwukrotnie z prawej strony zdarzenia **OnKeyDown** (druga kolumna), co spowoduje wygenerowanie procedury;
- Między słowami **begin** i **end** wpisz fragment kodu, uruchamiany w momencie naciśnięcia klawisz ENTER:

```
procedure TForm1.ComboBox1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  // Wprowadzenie nowego ciągu znaków do listy przez naciśnięcie klawisza ENTER.
  if (Key = VK_RETURN) then
  begin
    // Wywołanie funkcji umożliwiającej dodanie nowego ciągu znaków.
    sbDodajClick(Sender);
  end;
end;
```

Każde naciśnięcie klawisza ENTER lub przycisku „+” spowoduje dodanie elementu do listy.

- Wybierz przycisk z napisem „-” i kliknij na nim dwukrotnie, co spowoduje wygenerowanie procedury. Między słowami **begin** i **end** wpisz kod odpowiedzialny za usunięcie elementu z listy:

```
procedure TForm1.sbUsunClick(Sender: TObject);
var
  numBtn: Integer;
begin
  // Usunięcie wybranego elementu z listy.

  // Application.MessageBox - wywołanie okna dialogowego.
  numBtn:= Application.MessageBox('Czy usunąć ten element z listy ?',
            PChar(Label1.Caption), MB_ICONQUESTION or MB_YESNO);
  if (numBtn = IDYES) then
  begin

    // Usunięcie zaznaczonego elementu z listy.
    ComboBox1.Items.Delete(ComboBox1.ItemIndex);

    ComboBox1.Items.SaveToFile('Lista.txt'); // Zapisanie listy do pliku.

    ComboBox1.Text:= ''; // Wyczyszczenie zawartości edytora.

    ComboBox1.SetFocus; // Ustawienie listy z edytorem jako aktywny.
  end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.9. Menu

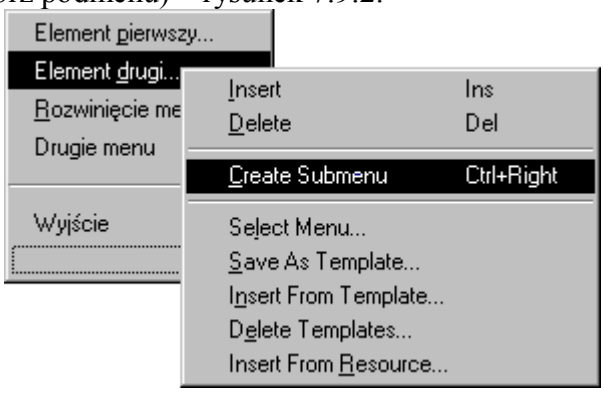
Napisz program, w którym jest możliwość przełączania między dwoma Menu. Poniższy rysunek przedstawia taki program.

Rysunek 7.9.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\MENU.

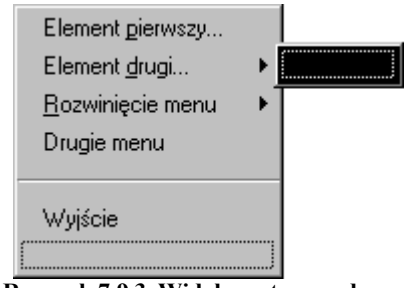
Opis komponentu:

Menu jest komponentem służącym do wywoływania funkcji programu uprzednio umieszczonych w menu. Na jednej formatce może być więcej niż jedno menu. Wybranie menu nas interesującego odbywa się za pomocą własności **Menu** obiektu formatki. Wyboru tego dokonujemy pisząc w programie linie **Menu:= MainMenu1**, lub **Menu:= MainMenu2** w zależności od tego, które menu ma być aktywne. W danej chwili jest widoczne tylko jedno menu. Komponent ten znajduje się na karcie **Standard** palety komponentów.
Menu może być rozwijane do podmenu. Tworzenie podmenu polega na wywołaniu funkcji **Create SubMenu** (Twórz podmenu). Aby element posiadał podmenu musimy go wybrać i prawym klawiszem myszy wywołać menu podręczne, z którego należy wybrać funkcję **Create SubMenu** (Twórz podmenu) – rysunek 7.9.2.



Rysunek 7.9.2. Widok wybranego polecenia Create SubMenu (Twórz podmenu)

Po wybraniu opcji **Create SubMenu** (Twórz podmenu) ukaże się nam puste podmenu – rysunek 7.9.3.



Rysunek 7.9.3. Widok pustego podmenu

Sposób wykonania:

- Wybierz dwa komponenty **MainMenu** z palety komponentów (karta **Standard**);
- Pierwsze menu opisz zgodnie z rysunkiem 7.9.1, oraz poszczególnym elementom menu nadaj nazwy:

  Nazwy poszczególnych elementów pierwszego menu są następujące:
  - PlikElementpierwszy – widoczny opis to „Element pierwszy...";
  - PlikElementdrugi – widoczny opis to „Element drugi...";
  - PlikRozwinieciemenu – widoczny opis to „Rozwinięcie menu";
  - PlikDrugiemenu – widoczny opis to „Drugie menu";
  - PlikWyjscie – widoczny opis to „Wyjście".

  Nazwy poszczególnych elementów podmenu „Rozwinięcie menu" są następujące:
  - PlikRozwMenuElement1 – widoczny opis to „Element 1...";
  - PlikRozwMenuElement2 – widoczny opis to „Element 2...".
- W drugim menu umieść pozycję **Menu pierwsze**, inne elementy menu są dowolne;
- Wybierz pierwszy element menu pierwszego i kliknij na nim. W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.PlikElementpierwszyClick(Sender: TObject);
begin
  // Wywołanie elementu pierwszego z menu głównego
  ShowMessage(PlikElementpierwszy.Caption);
end;
```

Z innymi elementami menu postępuj tak samo, prócz pozycji „Drugie menu" i pozycji „Wyjście".

- Wybierz pozycję menu o nazwie **Drugie menu** i kliknij na nim. W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.PlikDrugiemenuClick(Sender: TObject);
begin
  // Przełącza się na drugie menu
  Menu:= MainMenu2;
end;
```

- Wybierz pozycję menu o nazwie **Wyjście** i kliknij na nim. W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.PlikWyjcieClick(Sender: TObject);
begin
  // Wyjście z programu
  Close;
end;
```

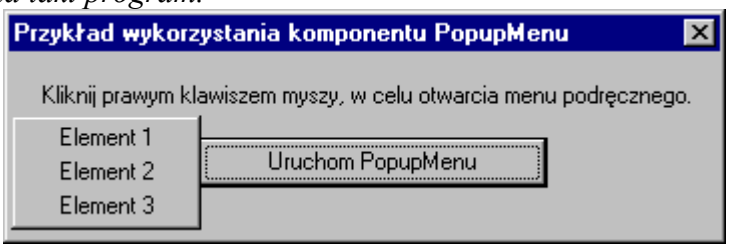
- Wybierz pozycję w menu drugim o nazwie **Menu pierwsze** i kliknij na nim. W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Menupierwsze1Click(Sender: TObject);
begin
  // Przełącza się na pierwsze menu
  Menu:= MainMenu1;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.10. PopupMenu

Wykonaj program, w którym opcje będą wybierane za pomocą menu podręcznego. Poniższy rysunek przedstawia taki program.



Rysunek 7.10.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\POPMENU.

Opis komponentu:

PopupMenu jest komponentem takim samym jak menu z tą różnicą, że można go wywołać za pomocą prawego klawisza myszy. **PopupMenu** można podłączyć do komponentów znajdujących się na formatce. Ponadto jest on niewidoczny w czasie działania programu jak np. menu główne. Dopiero użytkownik wywołując go powoduje ukazanie się tegoż menu podręcznego. Komponent ten znajduje się na karcie **Standard** palety komponentów.

Sposób wykonania:

- Wybierz dwa komponenty **PopupMenu** z palety komponentów (karta **Standard**);
- Opis elementów **PopupMenu** jest dowolny;
- Rozwiązanie i obsługa **PopupMenu** jest taka sama jak **MainMenu**;
- Wybierz klawisz **Button** (karta **Standard**);
- Kliknij dwukrotnie na formatce i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // Programowe podłączenie menu podręcznego do formatki.
  PopupMenu:= PopupMenu1;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Uruchom PopupMenu” i w wygenerowanej procedurze wpisz kod:

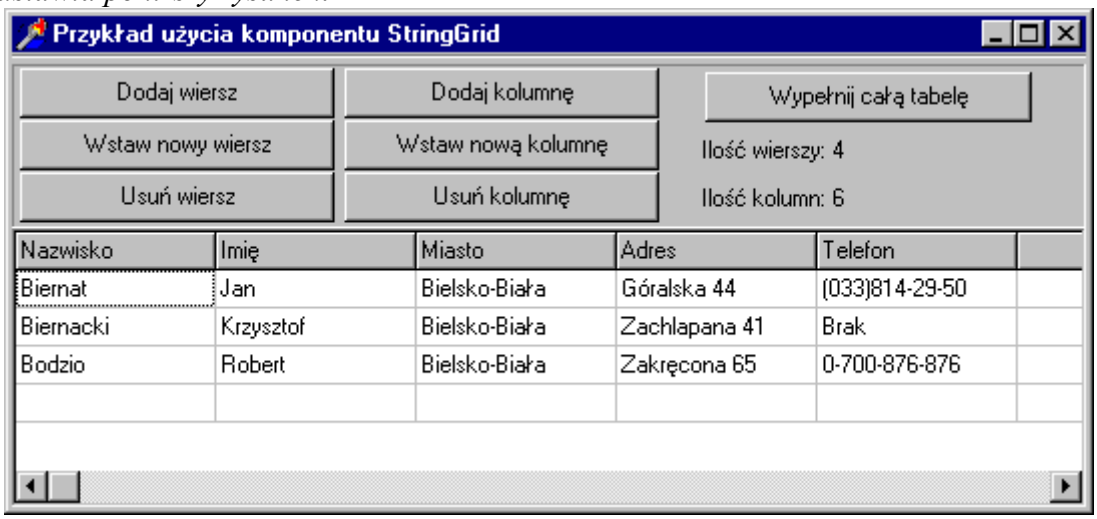
```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  // Uruchomienie menu podręcznego
  PopupMenu2.Popup(Left+Button1.Left+2, Top+Button1.Top-35);
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.11. StringGrid

Napisz program, który wypełni arkusz i będzie miał możliwość dodania wiersza lub kolumny oraz będzie miał możliwość wstawienia całego wiersza lub całej kolumny. Wygląd programu przedstawia poniższy rysunek.



Rysunek 7.11.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\STRGRID.

Opis komponentu:

StringGrid (arkusz pól edycyjnych) jest komponentem, który umożliwia wyświetlenie informacji w sposób tabelaryczny lub zrobienie prostego programu kalkulacyjnego. Komponent ten znajduje się na karcie **Standard** palety komponentów.

Sposób wykonania:

- Wybierz komponent **StringGrid** z palety komponentów (karta **Additional**);
- Wybierz kilka komponentów **Button** (karta **Standard**) i opisz je zgodnie z rysunkiem 7.11.1;
- Wybierz dwa komponenty **Label** z palety komponentów (karta **Standard**)
- Kliknij dwukrotnie na klawisz z napisem „Dodaj wiersz” i wpisz w wygenerowanej procedurze kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Dodanie nowego wiersza
  StringGrid1.RowCount:= StringGrid1.RowCount+1;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Wstaw nowy wiersz” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  AA, BB: Integer;
begin
  // Wstawia nowy wiersz

  // Przesunięcie (przez przekopiowanie) wierszy o jeden wiersz w dół
  for AA:= StringGrid1.RowCount-1 downto StringGrid1.Row do
    for BB:= 0 to StringGrid1.ColCount-1 do
      StringGrid1.Cells[BB, 1+AA]:= Trim(StringGrid1.Cells[BB, AA]);

  // Wyczyszczenie całego wiersza, przez co powstaje nowy pusty wiersz
  for AA:= 0 to StringGrid1.ColCount-1 do
    StringGrid1.Cells[AA, StringGrid1.Row]:= '';

  // Dodanie nowego wiersza
  StringGrid1.RowCount:= StringGrid1.RowCount+1;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Usuń wiersz” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button3Click(Sender: TObject);
var
  AA, BB: Integer;
begin
  // Usuwa wiersz

  // Czyści cały wiersz
  for AA:= 0 to StringGrid1.ColCount-1 do
    StringGrid1.Cells[AA, StringGrid1.Row]:= '';

  // Przesunięcie (przez przekopiowanie) wierszy o jeden wiersz wyżej
  for AA:= 0 to StringGrid1.RowCount-1 do
    for BB:= 0 to StringGrid1.ColCount-1 do
    begin
      StringGrid1.Cells[BB, StringGrid1.Row+AA]:= Trim(
              StringGrid1.Cells[BB, StringGrid1.Row+1+AA]);
    end;

  // Usunięcie wiersza
  StringGrid1.RowCount:= StringGrid1.RowCount-1;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Dodaj kolumnę” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  // Dodanie nowej kolumny
  StringGrid1.ColCount:= StringGrid1.ColCount+1;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Wstaw nową kolumnę” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button5Click(Sender: TObject);
var
  AA, BB: Integer;
begin
  // Wstawia nową kolumnę

  // Przesunięcie (przez przekopiowanie) kolumny o jedną kolumnę w prawą stronę
  for AA:= StringGrid1.ColCount-1 downto StringGrid1.Col do
    for BB:= 0 to StringGrid1.RowCount-1 do
      StringGrid1.Cells[1+AA, BB]:= Trim(StringGrid1.Cells[AA, BB]);

  // Wyczyszczenie całej kolumny
  for AA:= 0 to StringGrid1.RowCount-1 do
    StringGrid1.Cells[StringGrid1.Col, AA]:= '';

  // Dodanie nowej kolumny
  StringGrid1.ColCount:= StringGrid1.ColCount+1;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Usuń kolumnę” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button6Click(Sender: TObject);
var
  AA, BB: Integer;
begin
  // Usuwa kolumnę

  // Wyczyszczenie całej kolumny
  for AA:= 0 to StringGrid1.RowCount-1 do
    StringGrid1.Cells[StringGrid1.Col, AA]:= '';

  // Przesunięcie (przez przekopiowanie) kolumny o jedną kolumnę w lewą stronę
  for AA:= 0 to StringGrid1.ColCount-1 do
    for BB:= 0 to StringGrid1.RowCount-1 do
    begin
      StringGrid1.Cells[StringGrid1.Col+AA, BB]:= Trim(
                  StringGrid1.Cells[StringGrid1.Col+1+AA, BB]);
    end;
```

```
  // Usunięcie kolumny
  StringGrid1.ColCount:= StringGrid1.ColCount-1;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Wypełnij całą tabelę” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  // Wypełnia jeden wiersz w tabeli

  // Tytuły kolumn
  StringGrid1.Cells[0, 0]:= 'Nazwisko';
  StringGrid1.Cells[1, 0]:= 'Imię';
  StringGrid1.Cells[2, 0]:= 'Miasto';
  StringGrid1.Cells[3, 0]:= 'Adres';
  StringGrid1.Cells[4, 0]:= 'Telefon';

  // Dane
  StringGrid1.Cells[0, 1]:= 'Kowalski';
  StringGrid1.Cells[1, 1]:= 'Jan';
  StringGrid1.Cells[2, 1]:= 'Gdańsk';
  StringGrid1.Cells[3, 1]:= 'Błotna 22';
  StringGrid1.Cells[4, 1]:= '811-56-12';

  // Wyświetlenie ilości wierszy i kolumn
  Label1.Caption:= 'Ilość wierszy:'+CHR(32)+IntToStr(StringGrid1.RowCount-1);
  Label2.Caption:= 'Ilość kolumn:'+CHR(32)+IntToStr(StringGrid1.ColCount);
end;
```

- Wstaw dodatkowy klawisz i nazwij go „Zapisanie tabeli” i kliknij go dwukrotnie oraz wpisz kod w wygenerowanej procedurze:

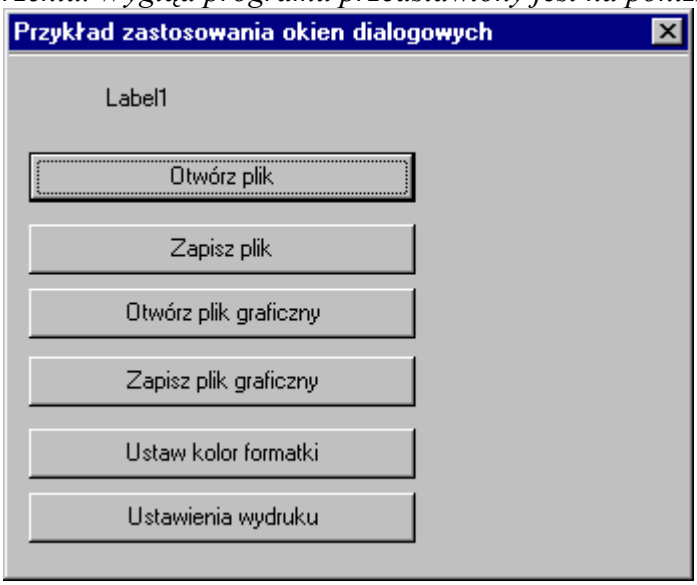
```
function TForm1.StringGridZapisz(txtFileName: String; numCol: Integer): Shortint;
var
  aCol, aRow: Integer; // Deklaracja zmiennych
        FT: TextFile; // Deklaracja zmiennej plikowej
begin
  // Zapisanie tabeli do pliku tekstowego
  StringGridZapisz:= -1;

  {
   if (Trim(txtFileName)<>'') then
  Sprawdzenie, czy została podana nazwa pliku. Jeżeli tak to zapisz (funkcja zwróci
wartość 1), w innym przypadku nie rób nic (funkcja zwróci wartość –1).
  }
  if (Trim(txtFileName)<>'') then
  begin
    AssignFile(FT, Trim(txtFileName)); // Powiązanie nazwy pliku z plikiem na dysku
```

```
    // Zmiana atrybutów pliku
    FileSetAttr(Trim(txtFileName), faArchive);

    Rewrite(FT); // Tworzenie i otwieranie pliku do zapisu

   // Zapisanie wypełnionych wierszy
    for aRow:= 0 to StringGrid1.RowCount-1 do
      for aCol:= 0 to StringGrid1.ColCount-1 do
        if (Trim(StringGrid1.Cells[numCol, aRow])<>'') then
          Writeln(FT, StringGrid1.Cells[aCol, aRow]);

    CloseFile(FT); // Zamknięcie pliku
    StringGridZapisz:= 1;
  end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.12. Okna dialogowe

Napisz program, który będzie prezentował okna dialogowe oraz wyświetlał pliki o odpowiednim rozszerzeniu. Wygląd programu przedstawiony jest na poniższym rysunku.



Rysunek 7.12.1. Wygląd programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\DIALOGI.

Opis okien dialogowych:
Okna dialogowe mają na celu ułatwić programiście tworzenie podstawowych funkcji aplikacji, takie jak: - otwarcie pliku, - zmiana kolorów, itp.
Okna dialogowe znajdują się na karcie **Dialogs** palety komponentów.
Tekst widoczny w oknach dialogowych zależy od zainstalowanej wersji językowej Windows'a.

Sposób wykonania:

- Wybierz kilka komponentów **Button** z palety komponentów (karta **Standard**);

Nazwy poszczególnych przycisków są następujące:

- bOtworzPlik – napis widoczny na przycisku to „Otwórz plik”;
- bZapiszPlik – napis widoczny na przycisku to „Zapisz plik”;
- bOtworzPlikGraficzny – napis widoczny na przycisku to „Otwórz plik graficzny”;
- bZapiszPlikGraficzny – napis widoczny na przycisku to „Zapisz plik graficzny”;
- bUstawKolorFormatki – napis widoczny na przycisku to „Ustaw kolor formatki”;
- bUstawieniaWydruku – napis widoczny na przycisku to „Ustawienia wydruku”.

♦ Wybierz następujące komponenty z karty **Dialogs**:

- **OpenDialog** (Otwórz plik);
- **SaveDialog** (Zapisz plik);
- **OpenPictureDialog** (Otwórz plik graficzny);
- **SavePictureDialog** (Zapisz plik graficzny);
- **ColorDialog** (Wybierz kolor);
- **PrinterSetupDialog** (Ustawienia wydruku).

♦ Kliknij dwukrotnie na przycisk z napisem „Otwórz plik” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bOtworzPlikClick(Sender: TObject);
begin
  // Otwiera okno dialogowe, które umożliwia nam wczytanie pliku
  {
   Nazwa "OpenDialog1" jest nazwą komponentu, którą można
   zmienić we właściwościach okna "Object Inspector".
  }

  OpenDialog1.FileName:= '';
  OpenDialog1.Execute;
  if (Trim(OpenDialog1.FileName)<>'') then
  begin
    // (W tym miejscu wpisujemy własny kod programu)
    Label1.Caption:= Trim(OpenDialog1.FileName);
    {
     Funkcja Trim() - likwiduje spacje po prawej
                      jak i po lewej stronie napisu
                      lub zmiennej
    }
  end;
end;
```

♦ Kliknij dwukrotnie na przycisk z napisem „Zapisz plik” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bZapiszPlikClick(Sender: TObject);
begin
  // Zapisuje plik
  SaveDialog1.FileName:= '';
  SaveDialog1.Execute;
  if (Trim(SaveDialog1.FileName)<>'') then
  begin
    // (W tym miejscu wpisujemy własny kod programu)
    Label1.Caption:= Trim(SaveDialog1.FileName);
  end;
end;
```

- Kliknij dwukrotnie na przycisk z napisem „Otwórz plik graficzny” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bOtworzPlikGraficznyClick(Sender: TObject);
begin
  // Otwiera okienko do wybrania pliku graficznego
  OpenPictureDialog1.FileName:='';
  OpenPictureDialog1.Execute;
  if (Trim(OpenPictureDialog1.FileName)<>'') then
  begin
    // (W tym miejscu wpisujemy własny kod programu)
    Label1.Caption:= Trim(OpenPictureDialog1.FileName);
  end;
end;
```

- Kliknij dwukrotnie na przycisk z napisem „Zapisz plik graficzny” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bZapiszPlikGraficznyClick(Sender: TObject);
begin
  // Zapisz plik graficzny
  SavePictureDialog1.FileName:='';
  SavePictureDialog1.Execute;
  if (Trim(SavePictureDialog1.FileName)<>'') then
  begin
    // (W tym miejscu wpisujemy własny kod programu)
    Label1.Caption:= Trim(SavePictureDialog1.FileName);
  end;
end;
```

- Kliknij dwukrotnie na przycisk z napisem „Ustaw kolor formatki” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bUstawKolorFormatkiClick(Sender: TObject);
begin
  // Ustawia kolor formatki
  if ColorDialog1.Execute then
  begin
```

```
    Form1.Color:= ColorDialog1.Color; // Ustawia kolor formatki
    Label1.Caption:= IntToStr(ColorDialog1.Color); // Podaje wartość koloru
  end;
end;
```

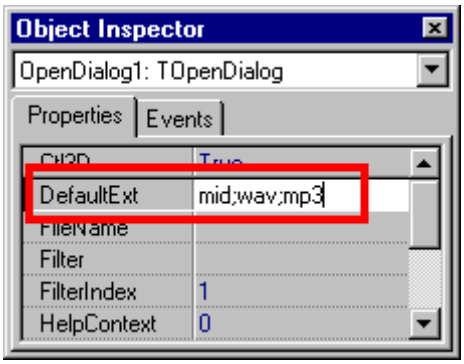
- Kliknij dwukrotnie na przycisk z napisem „Ustawienia wydruku” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bUstawieniaWydrukuClick(Sender: TObject);
begin
  // Otwiera okienko z ustawieniami wydruku
  PrinterSetupDialog1.Execute;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

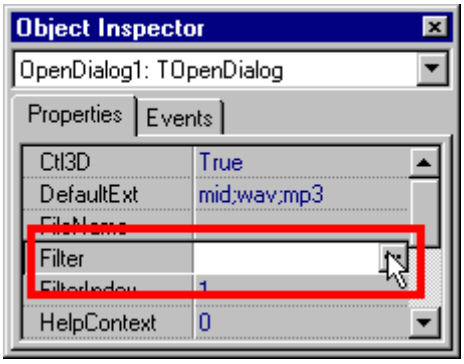
Okna dialogowe służące do wyświetlania plików, mają możliwość określenia rodzaju wyświetlanych plików. W celu zdefiniowania rodzaju wyświetlanego pliku wykonaj następujące czynności:

- Kliknij na komponent **OpenDialog1**, w celu zaznaczenia go;
- Naciśnij klawisz funkcyjny **F11**, w celu przejścia do okna **Object Inspector** (Inspektora obiektów);
- Będąc na zakładce **Properties** (Właściwości) wybierz następujące właściwości:
  - **DefaultExt** (Domyślne rozszerzenie) i wpisz np. „mid;wav;mp3” – rysunek 7.12.2;



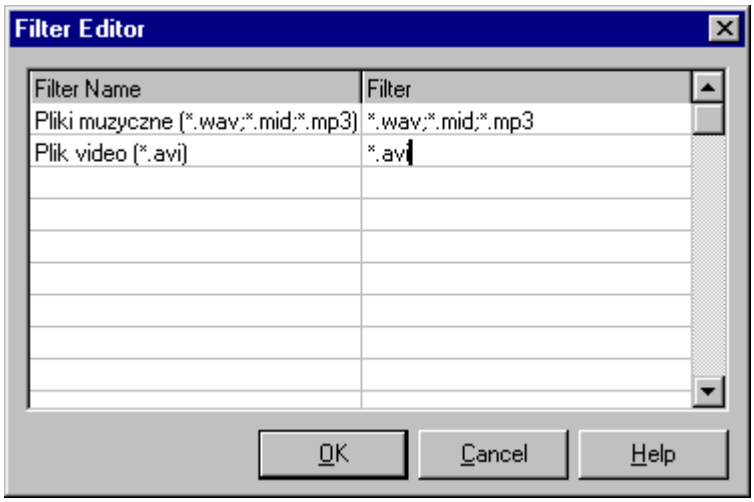
Rysunek 7.12.2. Widok wpisanego rozszerzenia

  - **Filter** (Filtr) i kliknij na klawisz ..., który znajduje się z prawej strony – rysunek 7.12.3;



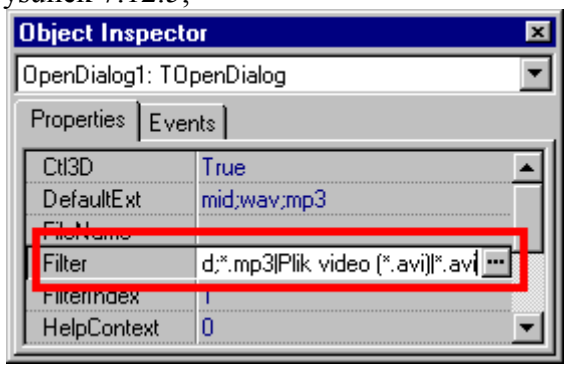
Rysunek 7.12.3. Widok wskazanego przycisku z trzema kropkami

  - W oknie **Filter Editor** (Edytor filtrów) wpisz odpowiednie rozszerzenia plików oraz ich nazwy – rysunek 7.12.4;



Rysunek 7.12.4. Widok okna Edytora Filtrów

- Po wpisaniu kliknij na klawisz **OK**, co spowoduje wypełnienie właściwości **Filter** (Filtr) – rysunek 7.12.5;



Rysunek 7.12.5. Widok wypełnionego pola właściwości Filter

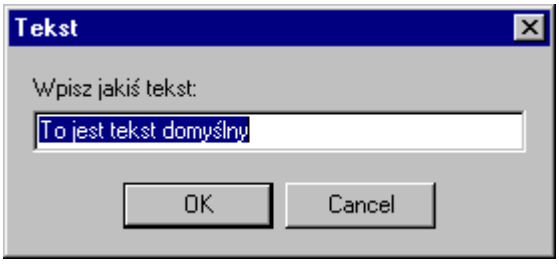
Rozszerzenie nazw pliku jest to trzyliterowe (przeważnie) rozszerzenie nazwy pliku, które oddzielone jest od głównej nazwy pliku kropką (np. „nazwa.doc”). Rozszerzenia nazw plików używane są przez różne programy, w celu łatwiejszej pracy z nimi.

Ćwiczenie 7.13. funkcja InputQuery

Napisz program, który umożliwi wpisanie tekstu przy pomocy polecenia InputQuery.

Przykład znajduje się w katalogu D7CW\INPUTQUERY.

Opis:
Okno **InputQuery** służy do wprowadzania danych. Wygląd tego okna ilustruje rysunek 7.13.1.



Rysunek 7.13.1. Widok okna InputQuery służącego do wprowadzania danych

Sposób wykonania:

- Wybierz komponent **Button** z palety komponentów (karta **Standard**);
- Kliknij dwukrotnie na ten klawisz i w wygenerowanej procedurze **OnClick** wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  InputQueryOK :Boolean; // Zadeklarowanie zmiennej logicznej
  NowyTekst :String; // Zadeklarowanie zmiennej tekstowej
begin
  // Wywołanie okna edycyjnego 'InputQuery'

  NowyTekst:= ''; // Wyczyszczenie zmiennej

  NowyTekst:= 'To jest tekst domyślny'; // Przypisanie zmiennej tekstu

  {
   Przypisanie wartości TRUE(PRAWDA) lub FALSE(FAŁSZ) przez
   funkcję "InputQuery" zmiennej "InputQueryOK".
  }
  InputQueryOK:= InputQuery('Tekst', 'Wpisz jakiś tekst:', NowyTekst);

  {
   "if (InputQueryOK = TRUE) then"
   Jeżeli funkcja "InputQuery" zwróci wartość TRUE(PRAWDA), to
   spełniony będzie warunek "InputQueryOK = TRUE", a co za
   tym idzie, nastąpi wykonanie instrukcji po słowie THEN.

   ==============================================

   "Button1.Caption:= NowyTekst"
   Przypisanie przyciskowi tekstu umieszczonemu w zmiennej NowyTekst
  }
  if (InputQueryOK = TRUE) then Button1.Caption:= NowyTekst;


  {
   InputQuery - Funkcja wprowadza do programu tekst, wprowadzony
            przez użytkownika.
            Wprowadzenie tekstu następuje, w momencie
            naciśnięcia klawisza ENTER lub kliknięcia na
            klawisz z napisem OK.
            W innym przypadku, tekst nie będzie
            wprowadzony do programu.

   ==============================================

   InputQuery(Nazwa_Okna,
          Tekst_Opisujący_Edytorek,
          Tekst_Domyslny);

   Nazwa_Okna - W tym parametrze wpisujemy nazwę okna,
```

która zostanie wyświetlona na pasku
tytułowym, np. "Tekst".

Tekst_Opisujący_Edytorek - W tym parametrze wpisujemy
tekst, który ukaże się nad
polem edycyjnym,
np. "Wpisz jakiś tekst:".

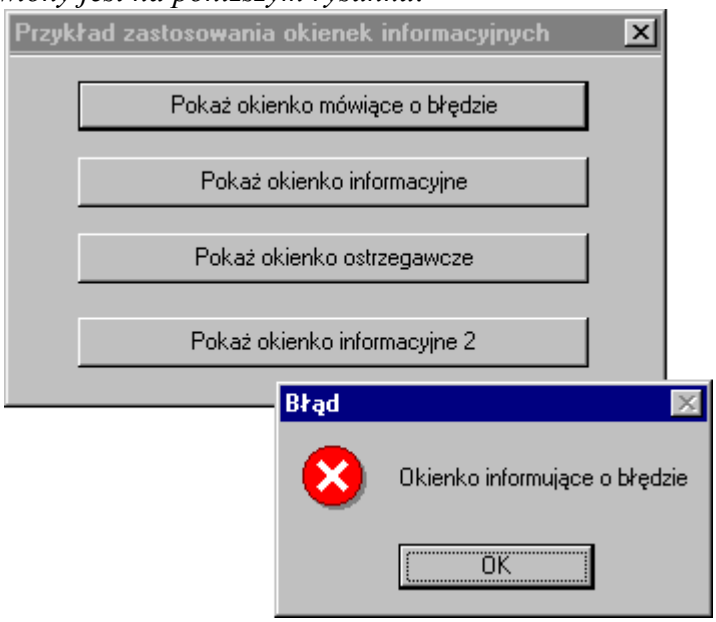
Tekst_Domyślny - W tym parametrze znajduje się zmienna (np. NowyTekst),
do której przypisany jest tekst domyślny.
Tekst ten jest wyświetlany w polu edycyjnym,
po uruchomieniu funkcji InputQuery.

}
end;

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.14. Okna informacyjne

Napisz program, który będzie prezentował rodzaje okien informacyjnych. Wygląd takiego programu przedstawiony jest na poniższym rysunku.

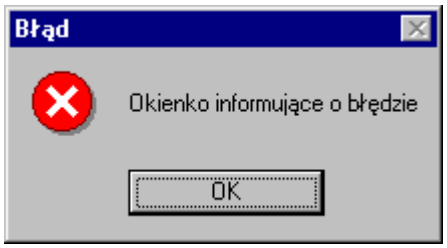


Rysunek 7.14.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\OKNAINF.

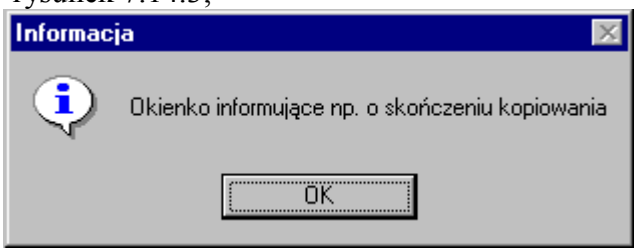
Opis okien informacyjnych:
Informacja wyświetlana jest za pomocą okien informacyjnych, które dzielą się na okna:

- Informujące o błędzie – rysunek 7.14.2;



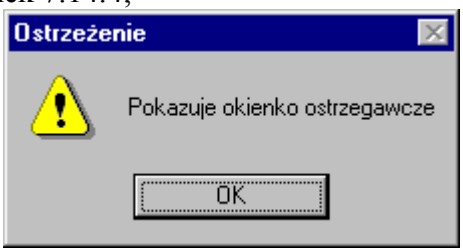
Rysunek 7.14.2. Widok okna informującego o błędzie

- Informacyjne – rysunek 7.14.3;



Rysunek 7.14.3. Widok okna informacyjnego

- Ostrzegawcze – rysunek 7.14.4;



Rysunek 7.14.4. Widok okna ostrzegawczego

Sposób wykonania:

- Wybierz kilka komponentów **Button** z palety komponentów (karta **Standard**); Nazwy poszczególnych przycisków są następujące:
  - bError – napis widniejący na przycisku to „Pokaż okienko mówiące o błędzie”;
  - bInformation – napis widniejący na przycisku to „Pokaż okienko informacyjne”;
  - bWarning – napis widniejący na przycisku to „Pokaż okienko ostrzegawcze”;
  - bInfo – napis widniejący na przycisku to „Pokaż okienko informacyjne 2”.
- Kliknij na przycisk z napisem **Pokaż okno mówiące o błędzie** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bErrorClick(Sender: TObject);
begin
  // Okno informujące o błędzie
  Application.MessageBox('Okno informujące o błędzie',
            'Błąd', MB_ICONERROR or MB_OK);
  {
  Application.MessageBox(Informacja,
            Tytuł_Okna, Rodzaj_Rysunku or Jaki_Klawisz);
  }
end;
```

- Kliknij dwukrotnie na przycisk z napisem **Pokaż okno informacyjne** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bInformationClick(Sender: TObject);
begin
  // Okno informujące np. o skończeniu kopiowania
  Application.MessageBox('Okno informujące o skończeniu kopiowania',
              'Informacja', MB_ICONINFORMATION or MB_OK);
end;
```

- Kliknij dwukrotnie na przycisk z napisem **Pokaż okno ostrzegawcze** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bWarningClick(Sender: TObject);
begin
  // Okienko ostrzegawcze np. o wystąpieniu jakiegoś błędu
  Application.MessageBox('Pokazuje okno ostrzegawcze',
              'Ostrzeżenie', MB_ICONWARNING or MB_OK);
end;
```

- Kliknij na przycisk z napisem **Pokaż okno informacyjne 2** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bInfoClick(Sender: TObject);
begin
  // Pokaż okno informacyjne
  ShowMessage('Okno ostrzegawcze !!');
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.15. ScrollBox i Image

Napisz program, który będzie umożliwiał wczytanie rysunku oraz przewijanie go na ograniczonym obszarze. Wygląd programu przedstawia poniższy rysunek.

Rysunek 7.15.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\IMAGE.

Opis komponentu:

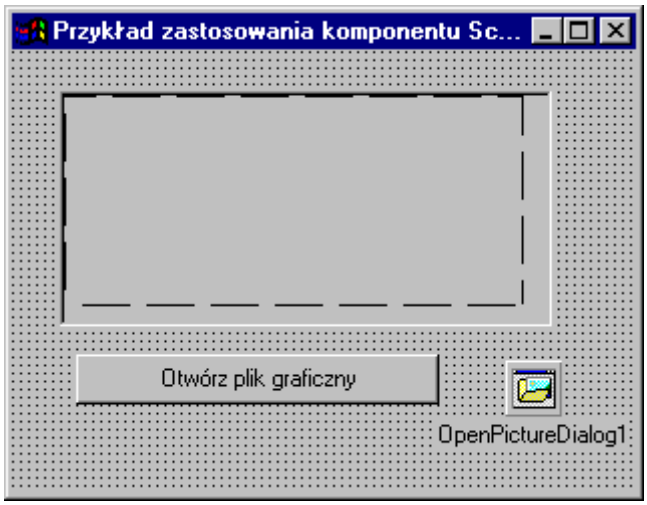
ScrollBox jest komponentem, który umożliwia nam przewijanie dużej ilości komponentów. Przydatne jest to w momencie, gdy ilość komponentów nie pozwala nam na umieszczenie ich na formatce. Komponent ten znajduje się na karcie **Additional** palety komponentów.

Image jest komponentem, który służy do wyświetlania grafiki na ekranie. **Image** znajduje się na karcie **Additional** palety komponentów.

Button jest komponentem, który znajduje się na karcie **Standard** palety komponentów. Służy on do uruchomienia jakiegoś zadania, w wyniku naciśnięcia tegoż przycisku.

Sposób wykonania:

- Wybierz komponent **OpenPictureDialog** (karta **Dialogs**);
- Wybierz komponent **ScrollBox** z palety komponentów (karta **Additional**);
- Wybierz komponent **Image** z palety komponentów (karta **Additional**) i kliknij na komponent **ScrollBox**, który wcześniej został umieszczony na formatce. Spowoduje to, że komponent **Image** nie wyjdzie poza granice komponentu **ScrollBox** - rysunek 7.15.2;



Rysunek 7.15.2. Widok nałożonego komponentu Image (przerywane linie) na komponent ScrollBox

- Wybierz przycisk **Button** (karta **Standard**) i nadaj mu opis „Otwórz plik graficzny”;
- Kliknij dwukrotnie lewym przyciskiem myszy na przycisku z napisem „Otwórz plik graficzny” i w wygenerowanej procedurze wpisz kod:

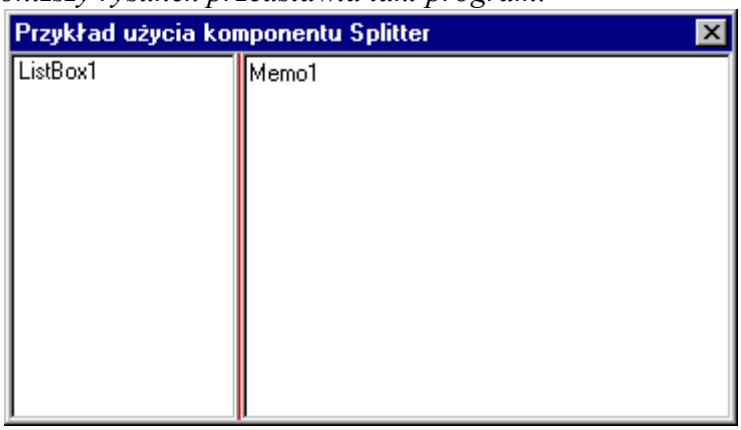
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Odczytanie pliku graficznego


  {
```

```
    Włączenie dopasowania się komponentu IMAGE1 do wielkości wczytanego
    rysunku (tzn. szerokość i wysokość komponentu IMAGE1 będzie taka
    sama jak wczytanego rysunku)
   }
   Image1.AutoSize:= TRUE;

   // Uruchomienie Okna Dialogowego, Który Umożliwia Wybór Pliku Graficznego
   OpenPictureDialog1.Execute;
   if (Trim(OpenPictureDialog1.FileName)<>'') then
   begin
     // Jeżeli Plik Zostanie Wybrany, To Wykonaj Poniższe Instrukcje

     // Wczytanie pliku graficznego
     Image1.Picture.LoadFromFile(Trim(OpenPictureDialog1.FileName));
   end;
 end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.16. Splitter

Wykonaj program, który będzie umożliwiał zmianę dwóch obszarów znajdujących się na jednej formatce. Poniższy rysunek przedstawia taki program.



Rysunek 7.16.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\SPLITTER.

Opis komponentu:

Splitter jest to komponent, który umożliwia zmianę obszaru prostokątnego podczas wykonywania programu. Przykładem wykorzystania takiego komponentu jest np. Eksplorator Windows, który jest podzielony na dwa panele. Granica tych dwóch paneli jest regulowana.

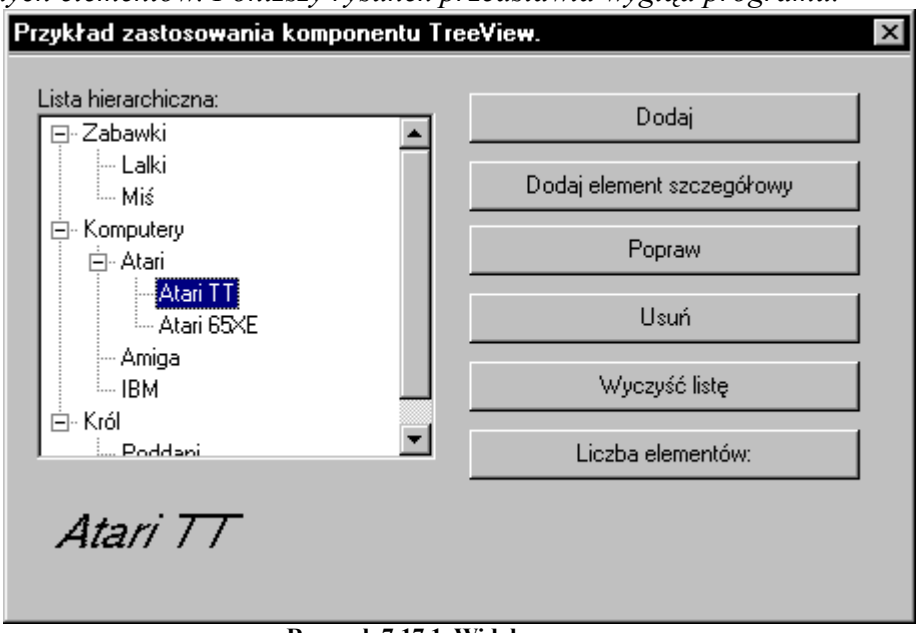
Sposób wykonania:

- Wybierz komponent **ListBox1** z palety komponentów (karta **Standard**)
- Zaznacz ten komponent i w oknie **Object Inspector** (Inspektora obiektów) wybierz zakładkę **Properties** (Właściwości). Następnie wybierz element listy **Align** i ustaw ją na wartość **alLeft**;

- Wybierz komponent **Splitter** (karta **Additional**) i we właściwości **Align** wybierz wartość **alLeft**;
- Wybierz komponent **Memo** (karta **Standard**) i we właściwości **Align** wybierz wartość **alClient**;
- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.17. TreeView

Napisz program, który przedstawia drzewo hierarchiczne pokazując przynależność poszczególnych elementów. Poniższy rysunek przedstawia wygląd programu.



Rysunek 7.17.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\TREEVIEW.

Opis komponentu:

TreeView jest komponentem, który umożliwia wyświetlenie informacji w postaci drzewa, które jest uporządkowane hierarchicznie. Komponent ten znajduje się na karcie **Win32** palety komponentów.

Przykład drzewa uporządkowanego hierarchicznie:

```
Król
  |
  -- Królowa
  |    |
  |    -- Służba
  |
Rycerz
  |
  -- Giermek
  |
```

-- itd.

Sposób wykonania:

- Wybierz komponent **TreeView** z palety komponentów (karta **Win32**);
- Wybierz kilka komponentów **Button** (karta **Standard**);
  Nazwy poszczególnych przycisków są następujące:
  - bDodaj – napis widoczny na przycisku to „Dodaj”;
  - bDodajElementSzczegolowy – napis widoczny na przycisku to „Dodaj element szczegółowy”;
  - bPopraw – napis widoczny na przycisku to „Popraw”;
  - bUsun – napis widoczny na przycisku to „Usuń”;
  - bWyczysc – napis widoczny na przycisku to „Wyczyść listę”;
  - bLiczbaElementw – napis widoczny na przycisku to „Liczba elementów”.
- Wybierz dwa komponenty **Label** (karta **Standard**) i umieść jeden nad, a drugi pod komponentem **TreeView**;
- Kliknij dwukrotnie na klawisz z napisem „Dodaj” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bDodajClick(Sender: TObject);
begin
  // Dodanie do listy tekstu
  TreeView1.Items.AddFirst(TreeView1.Selected,
                    InputBox('Dodaj', 'Tekst:', 'Pozycja'));
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Dodaj element szczegółowy” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bDodajElementSzczegowyClick(Sender: TObject);
begin
  // Dodaje element szczegółowy
  TreeView1.Items.AddChildFirst(TreeView1.Selected,
                    InputBox('Dodaj', 'Tekst:', 'Pozycja'));
  {
   TreeView1.Items.AddChildFirst - dodanie pozycji do listy lub
                        dodanie 'Elementu2' do 'Elementu1'
                        uprzednio zaznaczając 'Elememt1'.
                    Przez co tworzy się hierarchia.
  }
  // InputBox('Dodaj', 'Tekst:', 'Pozycja') - otwiera okno edycyjne
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Popraw” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bPoprawClick(Sender: TObject);
begin
  // Poprawia zaznaczony tekst
```

```
  TreeView1.Items.Insert(TreeView1.Selected,
                    InputBox('Dodaj', 'Tekst:', TreeView1.Selected.Text));
  // TreeView1.Items.Insert - Wstawia nowy tekst na pozycję elementu zaznaczonego

  TreeView1.Items.Delete(TreeView1.Selected); // Usuwa zaznaczony tekst
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Usuń” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bUsunClick(Sender: TObject);
begin
  // Usuwa zaznaczony tekst
  TreeView1.Items.Delete(TreeView1.Selected);
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Wyczyść listę” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bWyczyscClick(Sender: TObject);
begin
  // Wyczyszczenie zawartości listy
  TreeView1.Items.Clear;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Liczba elementów” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bLiczbaElementwClick(Sender: TObject);
begin
  // Podaje liczbę elementów
  bLiczbaElementw.Caption:= 'Liczba elementów: '+IntToStr(TreeView1.Items.Count);
end;
```

- W celu wyświetlenia w komponencie **Label** nazwy zaznaczonego elementu podczas poruszania się po liście, trzeba wykonać następujące czynności:
  - Zaznaczyć komponent **TreeView** (uprzednio trzeba go umieścić na formatce), przez kliknięcie na nim;
  - Nacisnąć klawisz funkcyjny **F11**, co spowoduje przyjście do okna **Object Inspector** (Inspektora obiektów);
  - Kliknąć na zakładkę **Events** (Zdarzenia);
  - Kliknąć z prawej strony napisu **OnChange**;
  - W wygenerowanej procedurze wpisać kod:

```
procedure TForm1.TreeView1Change(Sender: TObject; Node: TTreeNode);
begin
  // Wyświetla zaznaczony tekst

  {
```

```
  Zmiana koloru wyświetlanego napisu w zależności od
  spełnionego warunku, w innym przypadku kolor będzie czarny.
 }
 if (TreeView1.Selected.Text = 'Atari 65XE') then
   Label2.Font.Color:= clBlue
 else
 if (TreeView1.Selected.Text = 'IBM') then
   Label2.Font.Color:= clGreen
 else
 if (TreeView1.Selected.Text = 'Lalki') then
   Label2.Font.Color:= clRed
 else
   Label2.Font.Color:= clBlack;

 // Wyświetla zaznaczony tekst
 Label2.Caption:= TreeView1.Selected.Text;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.18. Wewnętrzna lista

Napisz program, który utworzy wewnętrzną listę, doda do niej elementy i na koniec zapisze listę do pliku.

Przedstawiony przykład znajduje się w katalogu D7CW\LISTA.

Opis wewnętrznej listy:
Lista wewnętrzna umożliwia nam przechowywanie elementów listy, bez korzystania z komponentów takich jak: **ComboBox**, **ListBox** czy **CheckListBox**. Lista ta w czasie działania aplikacji jest niewidoczna dla użytkownika.

Sposób wykonania:

- W celu stworzenia listy musisz ją zadeklarować. Zrób to przez wstawienie do sekcji **Private** (Prywatny) deklaracji „List: TStringList;”. Przykład wstawienia deklaracji jest przedstawiony poniżej.

```
private
  { Private declarations }
  List: TStringList; // Obiekt reprezentujący naszą listę.
```

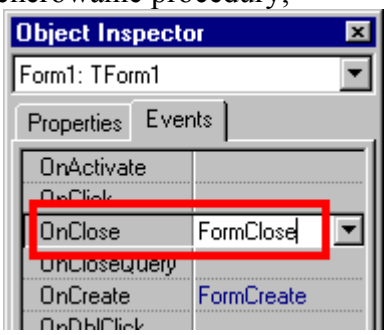
- Następnie kliknij dwukrotnie na formatce, co spowoduje wygenerowanie zdarzenia **OnCreate**. W wygenerowanej procedurze stwórz listę poprzez wpisanie linii „List:= TStringList.Create;”. Przykład jest zamieszczony poniżej.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  List:= TStringList.Create; // Dynamiczne tworzenie listy
end;
```

- Wybierz komponent **Button** z palety komponentów (karta **Standard**) i nadaj mu nazwę „bDodajDoListyElementyZapisujacDoPliku” oraz opis (widoczny na przycisku) „Dodaj do listy elementy i zapisz do pliku”;
- Kliknij dwukrotnie na nim i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bDodajDoListyElementyZapisujacDoPlikuClick(Sender: TObject);
begin
  // Dodaje elementy do listy i zapisuje ją do pliku

  List.Clear; // Czyszczenie listy

  List.Add('Spectrum'); // Dodanie do listy
  List.Add('Amiga');
  List.Add('IBM');
  List.Add('Atari');
  List.Add('Żyrafa');
  List.Add('Star Wars');
  List.Add('Cry');
  List.Add('CPU');
  List.Add('Komputer');

  List.Sort; // Włącza sortowanie listy

  Label1.Caption:= Trim(List.Strings[0]); // Odczytanie elementu z listy o podanym
numerze.

  List.SaveToFile('Lista.txt'); // Zapisanie własnej listy do pliku.
end;
```

- Kliknij na formatce;
- Naciśnij klawisz funkcyjny **F11**, co spowoduje przejście do okna **Object Inspector** (Inspektora obiektów);
- Kliknij na zakładce **Events** (Zdarzenia);
- Kliknij dwukrotnie z prawej strony zdarzenia **OnClose** (druga kolumna) – rysunek 7.18.1, co spowoduje wygenerowanie procedury;



Rysunek 7.18.1. Widok nazwy wygenerowanej procedury OnClose

- W wygenerowanej procedurze **FormClose** wpisz linię przedstawioną poniżej:

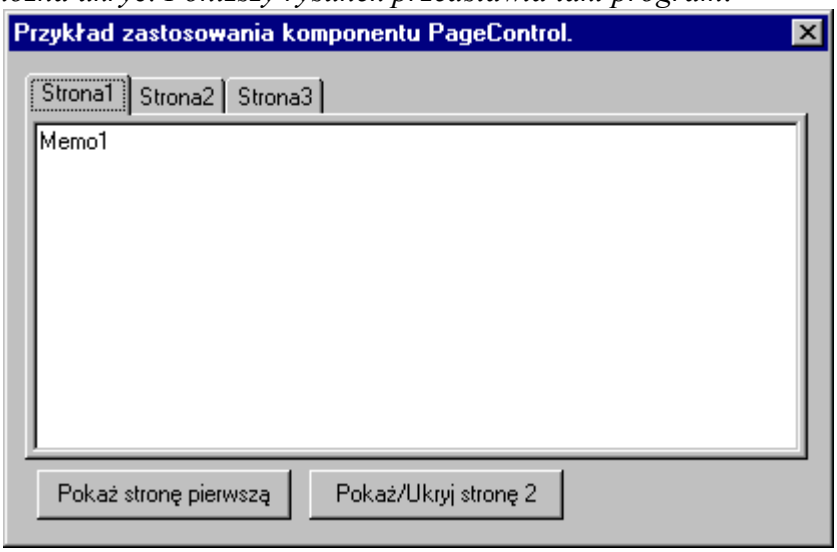
```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  List.Destroy; // Usunięcie listy z pamięci
```

end;

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.19. PageControl (Zakładki)

Napisz program, w którym komponenty będą ułożone na 3-ch różnych zakładkach. Zakładkę drugą będzie można ukryć. Poniższy rysunek przedstawia taki program.



Rysunek 7.19.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\PAGECONTROL.

Opis komponentu:

Komponent **PageControl** umożliwia tworzenie zakładek. Na każdej zakładce mogą znajdować się związane tematycznie komponenty. Przechodzenie między zakładkami jest możliwe za pomocą kombinacji klawiszy **CTRL+TAB** - do przodu lub **CTRL+SHIFT+TAB** - do tyłu.

Sposób wykonania:

- Wybierz dwa komponenty **PageControl** z palety komponentów (karta **Win32**);
- Stwórz trzy zakładki, wykonując następujące kroki:
  - Kliknij prawym klawiszem myszy na komponent **PageControl**, co spowoduje wyświetlenie menu podręcznego;
  - Wybierz polecenie **New Page** (Nowa strona), co spowoduje stworzenie nowej zakładki;
  - Wykonuj te same czynności, aż do osiągnięcia trzech zakładek.
- Nadaj tym zakładkom opisy zgodnie z rysunkiem 7.19.1, wykonując następujące kroki:
  - Kliknij lewym klawiszem myszy na pierwszą zakładkę;
  - Naciśnij klawisz funkcyjny **F11**, co spowoduje przejście do okna **Object Inspector** (Inspektora obiektów);
  - Na zakładce Properties (Właściwości) wybierz właściwość **Caption** i wpisz nazwę wybranej zakładce;
  - Z innymi zakładkami postępuj tak samo.
- Na każdej zakładce umieść kilka dowolnych komponentów;

- Wybierz dwa komponenty **Button** (karta **Standard**) i umieść je na formatce oraz opisz zgodnie z rysunkiem 7.19.1;
- Kliknij dwukrotnie na klawisz z napisem „Pokaż stronę pierwszą” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Pokaż stronę pierwszą
  PageControl1.ActivePage:= TabSheet1;
  {
   TabSheet1 - Nazwa strony pierwszej
  }
end;
```

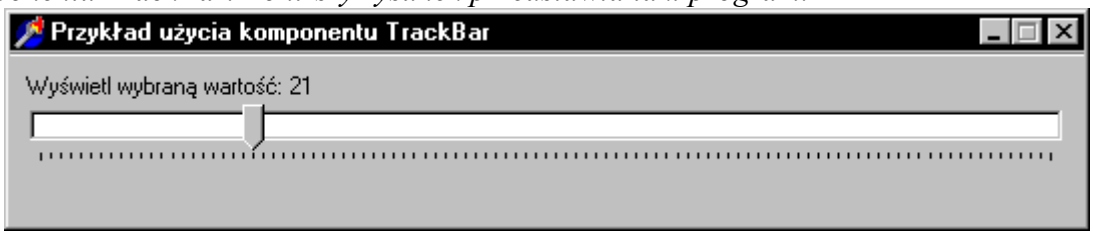
- Kliknij dwukrotnie na klawisz z napisem „Pokaż/Ukryj stronę 2” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  // Pokazuje/Ukrywa zakładkę (stronę) nr 2
  if (TabSheet2.TabVisible = TRUE) then
    TabSheet2.TabVisible:= FALSE
  else
    TabSheet2.TabVisible:= TRUE;
  {
   TabSheet2 - Nazwa strony drugiej
  }
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.20. TrackBar

Wykonaj program pokazujący zmieniającą się wartość pod wpływem poruszania wskaźnikiem komponentu TrackBar. Poniższy rysunek przedstawia taki program.



Rysunek 7.1.7.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\TRACKBAR.

Opis komponentu:

TrackBar (suwak) jest komponentem służącym do zmiany wartości (np. zmiany rozmiaru ekranu). **TrackBar** znajduje się na karcie **Win32** palety komponentów.

Sposób wykonania:

- Wybierz komponent **Label** z palety komponentów (karta **Standard**);
- Wybierz komponent **TrackBar** (karta **Win32**);
- Kliknij na komponent **TrackBar**, co spowoduje jego zaznaczenie;
- Naciśnij klawisz funkcyjny **F11**, co spowoduje przejście do okna **Object Inspector** (Inspektora obiektów);
- Będąc na zakładce **Properties** (Właściwości), wybierz **Max** i nadaj jej wartość 100;
- Kliknij dwukrotnie na komponent **TrackBar** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  // Wykonuje poniższe instrukcje w momencie ruszania wskaźnikiem.

  // Label1.Caption - Wyświetla wybraną wartość.
  Label1.Caption:= 'Wyświetl wybraną wartość: '+IntToStr(TrackBar1.Position);

{
   Label1.Caption - ten komponent służy do wyświetlania tekstu
   IntToStr('123') - ta funkcja wykonuje konwersję liczby na tekst
  }
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.21. ProgressBar i StatusBar

Napisz program, który będzie wykonywał pętle FOR..TO..DO i pokazywał postęp wykonywanej pracy.

Przykład znajduje się w katalogu D7CW\KOMPONENT\STBAR.

Opis komponentu:

ProgressBar jest komponentem, który umożliwia prezentację postępu wykonywanego zadania (np. zapisywania danych do pliku). Znajduje się on na karcie **Win32** palety komponentów.

StatusBar jest komponentem, który umożliwia wyświetlanie komunikatów. Umieszczany jest on na formatce w dolnej części. Znajduje się on na karcie **Win32** palety komponentów. W celu wyświetlenia tekstu informacyjnego w jednym panelu należy właściwość **SimplePanel** ustawić na wartość TRUE w oknie Inspektora Obiektów.
W innym przypadku jest możliwość podawania kilku informacji w jednym panelu równocześnie.

Sposób wykonania:

- Wybierz komponent **ProgressBar** z palety komponentów (karta **Win32**);
- Wybierz komponent **StatusBar** z palety komponentów (karta **Win32**);

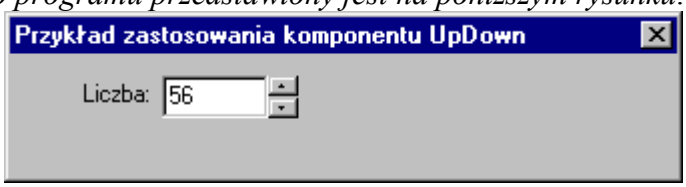
- Naciśnij klawisz funkcyjny **F11**, co spowoduje przejście do okna **Object Inspector** (Inspektora obiektów);
- Będąc na zakładce **Properties** (Właściwości), wybierz właściwość **SimplePanel** i ustaw jej wartość na TRUE;
- Następnie wybierz komponent **Button** (karta **Standard**) i nadaj mu nazwę „bDrugaInformacja" oraz opis (widoczny na przycisku) „Druga informacja";
- Po umieszczeniu go na formatce kliknij na nim dwukrotnie i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bDrugaInformacjaClick(Sender: TObject);
var
  TT: Integer;
begin
  // Druga informacja

  // Wyświetla informacje
  StatusBar1.SimpleText:= 'Teraz widzisz komponent "ProgressBar" w działaniu...';


  ProgressBar1.Position:= 0; // Ustawienie początkowej wartości
  ProgressBar1.Max:= 9999; // Określenie maksymalnej dopuszczalnej wartości
  for TT:= 0 to 9999 do
    ProgressBar1.Position:= TT; // Wskazanie postępu pracy pętli For

  // Wyświetla informacje
  StatusBar1.SimpleText:= 'ProgressBar jest bardzo przydatnym komponentem !!!!';
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.22. UpDown

Wykonaj program, który będzie zwiększał lub zmniejszał liczbę wyświetlaną w komponencie Edit. Wygląd takiego programu przedstawiony jest na poniższym rysunku.



Rysunek 7.22.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\UPDOWN.

Opis komponentu:

UpDown umożliwia zwiększanie lub zmniejszanie liczby wyświetlanej za pomocą komponentu np. EDIT. Komponent ten znajduje się na karcie **Win32** palety komponentów. Komponent **UpDown** można połączyć z innymi komponentami za pomocą właściwości **Associate** (Połącz z).

Wybrane właściwości komponentu **UpDown**:

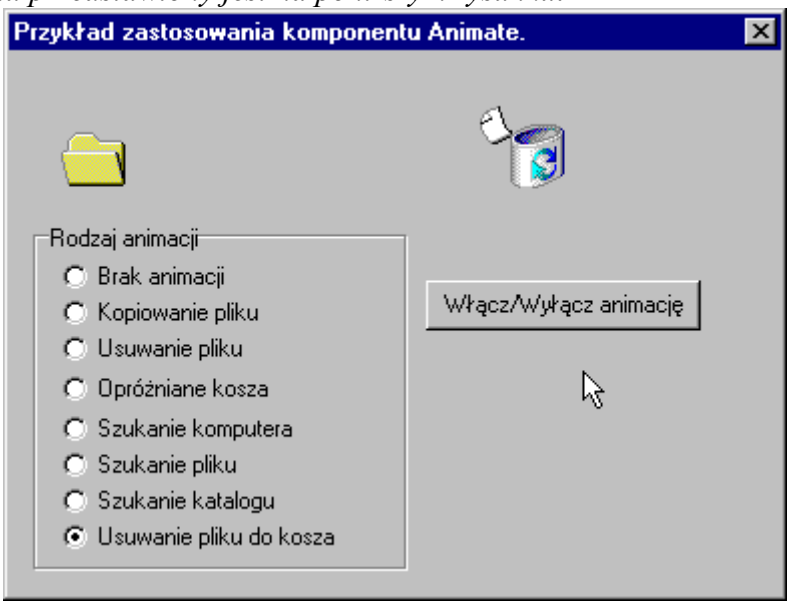
- **Increment** - w tej właściwości podajemy wartość o jaką będziemy zmniejszać lub zwiększać liczbę (np. 2, to liczba będzie zwiększana co 2 punkty. Domyślnie wpisana jest wartość 1);
- **ArrowKeys** - umożliwia zwiększanie i zmniejszanie za pomocą klawiatury (TRUE - włączona klawiatura; FALSE - wyłączona klawiatura);
- **Max** - określa maksymalną wartość;
- **Min** - określa minimalną wartość.

Sposób wykonania:

- Wybierz komponenty **Label** z palety komponentów (karta **Standard**);
- Wybierz komponent **Edit** (karta **Standard**);
- Wybierz komponent **UpDown** z palety komponentów (karta **Win32**);
- Kliknij na komponent **UpDown** ,w celu zaznaczenia go;
- Naciśnij klawisz funkcyjny **F11**, co spowoduje przejście do okna **Object Inspector** (Inspektora obiektów);
- Będąc na zakładce **Properties** (Właściwości), wybierz właściwość **Associate** komponentu **UpDown** i wybierz z listy rozwijanej **Associate** komponent **Edit1**;
- Po wybraniu komponentu **Edit** nastąpi połączenie z komponentem **UpDown**, który zostanie przysunięty do komponentu **Edit**;
- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.23. Animate - Standardowe animacje w systemie Windows

Napisz program, który zaprezentuje standardowe animacje występujące w systemie Windows. Wygląd programu przedstawiony jest na poniższym rysunku.



Rysunek 7.23.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\ANIMATE.

Opis komponentu:

Komponent **Animate** służy do odtwarzania standardowych animacji w systemie Windows, takich jak: - Kopiowanie plików; - Usuwanie plików; - Wyszukiwanie plików; - Opróżnianie kosza; - itp. Rodzaj animacji wybieramy za pomocą właściwości **CommonAVI** komponentu **Animate**.

Sposób wykonania:

- Wybierz komponent **Animate** z palety komponentów (karta **Win32**);
- Wybierz komponent **Button** (karta **Standard**) i opisz go jako „Włącz/Wyłącz animację”;
- Wybierz komponent **GroupBox** (karta **Standard**) i wykonaj następujące kroki:
  - Zaznacz go, przez kliknięcie na nim (jeżeli nie jest zaznaczony);
  - Naciśnij klawisz funkcyjny **F11**, w celu przejścia do okna **Object Inspector** (Inspektora obiektów);
  - Wybierz zakładkę **Properties** (Właściwości);
  - Wybierz właściwość **Caption** i wpisz tekst „Rodzaj animacji”.
- Wybierz komponent **RadioButton** i kliknij na komponent **GroupBox** opisany jako „Rodzaj animacji”;
- Ze wszystkimi komponentami **RadioButton** postępuj tak samo, przez rozmieszczenie ich na komponencie **GroupBox** – patrz rysunek 7.23.1;
- Zaznacz pierwszy komponent **RadioButton** i wykonaj następujące kroki:
  - Naciśnij klawisz funkcyjny **F11**, w celu przejścia do okna **Object Inspector** (Inspektora obiektów);
  - Wybierz zakładkę **Properties** (Właściwości);
  - Wybierz właściwość **Caption** i wpisz tekst „Brak animacji”;
  - W celu opisania następnych komponentów **RadioButton** postępuj jak poprzednio.
- Zaznacz komponent **RadioButton** opisany jako „Brak animacji”;
- Będąc na zakładce **Properties** (Właściwości) okna **Object Inspector** (Inspektora obiektów), wybierz właściwość **Checked** i ustaw ją na wartość TRUE (aktywny);
- Następnie kliknij dwukrotnie na komponent **RadioButton** opisany jako „Brak animacji” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  // Brak animacji
  Animate1.CommonAVI:= aviNone;
end;
```

- Kliknij dwukrotnie na drugim komponencie **RadioButton** opisanym jako „Kopiowanie pliku” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
  // Kopiowanie pliku
  Animate1.CommonAVI:= aviCopyFile;
end;
```

- Z następnymi komponentami **RadioButton** postępuj tak samo, jak przy drugim komponencie **RadioButton** opisanym jako „Kopiowanie pliku”, z tą różnicą, że właściwości **CommonAVI** będziesz przypisywał inne animacje (np. aviEmptyRecycle, aviFindFolder, itd.);
- Kliknij dwukrotnie na klawisz z napisem „Włącz/Wyłącz animację” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Włącza lub wyłącza animację
  if (Animate1.Active = TRUE) then
    Animate1.Active:= FALSE
  else
    Animate1.Active:= TRUE;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.24. Timer

Napisz program, który będzie pokazywał bieżący czas.

Przykład znajduje się w katalogu D7CW\KOMPONENT\TIMER.

Opis komponentu:

Timer jest komponentem, który służy do generowania zdarzeń w regularnych odstępach czasu. Wykorzystać go można np. do wyświetlania czasu. Znajduje się on na karcie **System** palety komponentów.

Sposób wykonania:

- Wybierz komponent **Label** z palety komponentów (karta **Standard**);
- Wybierz komponent **Timer** z palety komponentów (karta **System**);
- Kliknij dwukrotnie na komponencie **Timer** i w wygenerowanej procedurze **OnTimer** wpisz kod:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // Wyświetlenie godziny
  Label1.Caption:= TimeToStr(Time);
  // TimeToStr() - Dokonuje konwersji czasu na tekst
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.25. Grafika - Rysowanie na ograniczonym obszarze

Napisz program, który narysuje figury geometryczne, napisze tekst. Rysunki te mają być wykonane w ograniczonym obszarze – wykorzystaj komponent PaintBox. Wygląd programu przedstawiony jest na rysunku.

Rysunek 7.25.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\PAINTBOX.

Opis komponentu:

Komponent **PaintBox** pozwala na rysowanie grafiki na ograniczonym prostokątnym obszarze, dzięki temu programista nie musi kontrolować przekroczenia obszaru.

Sposób wykonania:

- Wybierz komponenty **PaintBox** z palety komponentów (karta **System**);
- Kliknij na komponent **PaintBox**, co spowoduje jego zaznaczenie;
- Naciśnij klawisz funkcyjny **F11**, co spowoduje przejście do okna **Object Inspector** (Inspektor Obiektów);
- Wybierz zakładkę **Events** (Zdarzenia);
- Będąc na tej zakładce wybierz zdarzenie **OnPaint** i kliknij dwukrotnie na pole obok tego napisu;
- W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  // PaintBox1Paint - Grafika jest odświeżana w przypadku zasłonięcia

  {
   Rysuj grafikę na ograniczonym obszarze za
   pomocą komponentu PainBox
  }

  // Wyczyszczenie obszaru rysowania.
  with PaintBox1.Canvas do
  begin
    // Ustalenie stylu wypełniania
    Brush.Style:= bsSolid;

    // Ustalenie koloru rysowania
    Brush.Color:= clWhite;
```

```
  // Wypełnienie aktywnego obszaru
  FillRect(ClipRect);
  {
   ClipRect - Reprezentuje bieżący prostokąt na
   którym są dokonywane operacje graficzne.
  }
end;


with PaintBox1.Canvas do
begin
  //-- Koło --
  // Ustalenie koloru jakim będzie narysowana figura.
  Pen.Color:= clBlack;

  // Ustalenie koloru jakim będzie wypełniona figura.
  Brush.Color:= clRed;

  // Narysowanie koła
  Ellipse(9, 9, 34, 34);


  //-- Kwadrat --
  // Ustalenie koloru jakim będzie narysowana figura.
  Pen.Color:= clBlack;

  // Ustalenie stylu pióra
  Pen.Style:= psDot;

  // Ustalenie koloru jakim będzie wypełniona figura.
  Brush.Color:= clRed;

  // Ustalenie stylu wypełniania
  Brush.Style:= bsDiagCross;

  // Narysowanie kwadratu
  Rectangle(39, 9, 66, 36);


  //-- Linia --
  MoveTo(66, 66); // Określenie początku linii
  LineTo(366, 66); // Narysowanie linii


  //-- Koło pochylone --
  // Ustalenie koloru jakim będzie narysowana figura.
  Pen.Color:= clBlack;

  // Ustalenie koloru jakim będzie wypełniona figura.
  Brush.Color:= clRed;
```

```
    // Ustalenie stylu wypełniania
    Brush.Style:= bsCross;

    // Narysowanie kwadratu
    Chord(199, 9, 77, 46, 8, 8, 8, 8);


    //-- Tekst --
    // Ustalenie czcionki tekstu
    Font.Name:= 'Arial';

    // Ustalenie koloru tekstu
    Font.Color:= clGreen;

    // Ustalenie stylu tekstu
    Font.Style:= [fsBold, fsItalic, fsUnderline];

    // Ustalenie wielkości tekstu
    Font.Size:= 33;

    // Wypisanie tekstu na ekranie
    TextOut(23, 88, 'To jest tekst.');
  end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.26. MediaPlayer

Napisz program do odtwarzania plików dźwiękowych.

Przykład znajduje się w katalogu D7CW\KOMPONENT\MEDIA.

Opis komponentu:

MediaPlayer jest komponentem służącym do odtwarzania plików muzycznych, muzyki z płyt CD, filmów i wiele innych. **MediaPlayer** znajduje się na karcie **System** palety komponentów.

Sposób wykonania:

- Wybierz komponent **MediaPlayer** z palety komponentów (karta **System**);
- Wybierz komponent **OpenDialog** (karta **Dialogs**);
- Wybierz komponent **Button** (karta **Standard**) i kliknij na nim dwukrotnie, a w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Otwórz plik muzyczny
```

```
  OpenDialog1.FileName:= '';
  OpenDialog1.Execute;
  if (Trim(OpenDialog1.FileName)<>'') then
  begin

    // Procedura otwierania i odtwarzania muzyki za pomocą komponentu MediaPlayer
    MediaPlayer1.Close;
    MediaPlayer1.FileName:= Trim(OpenDialog1.FileName);
    MediaPlayer1.Open;
    MediaPlayer1.Play;
  end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.27. Formatowanie liczby na format 1 234 567

Napisz program który sformatuje liczbę 12345678 na 12 345 678.

Przykład znajduje się w katalogu D7CW\WYS_LICZBE.

Sposób wykonania:

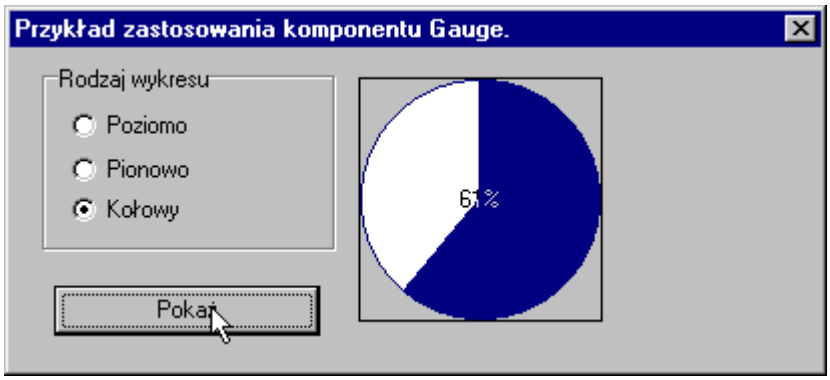
- Wybierz komponent **Label** z palety komponentów (zakładka **Standard**);
- Mając zaznaczony komponent **Label**, w oknie Inspektora Obiektów we właściwości **Caption** wpisz „Wprowadź liczbę:”;
- Wybierz komponent **Edit** z palety komponentów (zakładka **Standard**);
- Wybierz komponent **Button** (karta **Standard**) i kliknij na nim dwukrotnie, a w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Wyświetlenie w pasku tytułowym zawartości komponentu Edit.
  Caption:= FormatFloat('#,', StrToFloat(Edit1.Text));
  {
   FormatFloat('#,', Liczba)
   Funkcja służy do formatowania wprowadzonej liczby jako parametr "Liczba".
   Dzięki tej funkcji liczba jest wyświetlana w bardzo czytelny
   sposób, tj. 1 234 567.
  }
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.28. Gauge - Wykres

Napisz program ilustrujący przykładowe możliwości komponentu Gauge.
Przykładowy program jest przedstawiony na poniższym rysunku.



Rysunek 7.28.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\GAUGE.

Opis komponentu:
Komponent **Gauge** służy do pokazywania postępu wykonywanej pracy. Pasek postępu może być przedstawiony w pozycji poziomej i pionowej lub w postaci koła.

Sposób wykonania:

- Wybierz komponent **GroupBox** (karta **Standard**) i opisz go jako „Rodzaj wykresu”;
- Wybierz trzy komponenty **RadioButton** (karta **Standard**) i umieść je na komponencie **GroupBox** opisanym jako „Rodzaj wykresu” (dzięki temu, każde przesunięcie komponentu **GroupBox** spowoduje przesuwanie się również komponentów **RadioButton**);
- Opisz komponenty **RadioButton** – patrz rysunek 7.28.1;
- Wybierz komponent **Button** (karta **Standard**) i opisz go jako „Pokaż”;
- Wybierz komponent **Gauge** (karta **Sample**);
- Kliknij dwukrotnie na formie i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  {
   Tu są wpisywane instrukcje, które
   są wykonane w momencie tworzenia formatki.
  }


  //-- Inicjalizacja komponentu Gauge --

  // Określenie koloru tła
  Gauge1.BackColor:= clWhite;

  // Określenie koloru paska
  Gauge1.ForeColor:= clNavy;

  // Określenie minimalnej wartości
  Gauge1.MinValue:= 0;
```

```
  // Określenie maksymalnej wartości
  Gauge1.MaxValue:= 100;

  // Określenie ilości wykonanego zadania
  Gauge1.Progress:= 0;

  // Wywołanie pierwszej opcji „Poziomo”
  RadioButton1Click(Sender);
end;
```

- Kliknij dwukrotnie komponent **RadioButton** z napisem „Poziomo” i wpisz kod w wygenerowanej procedurze:

```
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  // Ustawienie wykresu w pozycji poziomej
  Gauge1.Kind:= gkHorizontalBar;
  Gauge1.Width:= 144; // Szerokość
  Gauge1.Height:= 22; // Wysokość

  // Określenie ilości wykonanego zadania
  Gauge1.Progress:= 0;

  // Uaktywnienie opcji
  RadioButton1.Checked:= TRUE;
end;
```

- Kliknij dwukrotnie komponent **RadioButton** z napisem „Pionowo” i wpisz kod w wygenerowanej procedurze:

```
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
  // Ustawienie wykresu w pozycji pionowej
  Gauge1.Kind:= gkVerticalBar;
  Gauge1.Width:= 22; // Szerokość
  Gauge1.Height:= 122; // Wysokość

  // Określenie ilości wykonanego zadania
  Gauge1.Progress:= 0;

  // Uaktywnienie opcji
  RadioButton2.Checked:= TRUE;
end;
```

- Kliknij dwukrotnie komponent **RadioButton** z napisem „Kołowy” i wpisz kod w wygenerowanej procedurze:

```
procedure TForm1.RadioButton3Click(Sender: TObject);
begin
  // Ustawienie wykresu w postaci koła
```

```
  Gauge1.Kind:= gkPie;
  Gauge1.Width:= 122; // Szerokość
  Gauge1.Height:= 122; // Wysokość

  // Określenie ilości wykonanego zadania
  Gauge1.Progress:= 0;

  // Uaktywnienie opcji
  RadioButton3.Checked:= TRUE;
end;
```

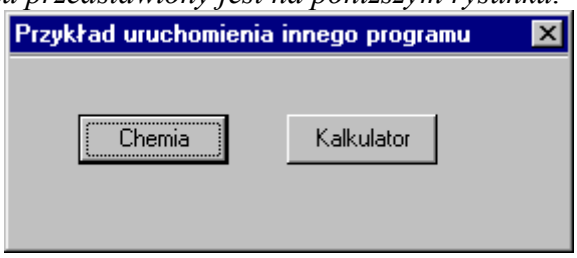
- Kliknij dwukrotnie przycisk z napisem „Pokaż” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TT: Integer; // Zadeklarowanie zmiennej "TT"
begin
  // Pokaż postęp wykonanej pracy pętli FOR
  for TT:= 0 to 99 do
    Gauge1.Progress:= 1+TT;
  // Pętla FOR zostanie wykonana 1000 razy
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.29. Uruchomienie innego programu z poziomu aplikacji

Wykonaj program, który będzie uruchamiał program np. KALKULATOR i CHEMIA. Wygląd takiego programu przedstawiony jest na poniższym rysunku.



Rysunek 7.29.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\RUNAPP.

Opis:
Możliwość uruchomienia innego programu (np. kalkulatora), z aktualnie działającej aplikacji jest dużym ułatwieniem. Pozwala to, na dołączenie do pisanego programu zewnętrznego programu (np. kalkulator).

Sposób wykonania:

- Wpisz nazwę biblioteki **ShellApi** w deklaracji **Uses**, np. **uses** ShellApi (bez tej deklaracji funkcja **ShellExecute** nie będzie działać);

- Wybierz dwa komponenty **Button** z palety komponentów (karta **Standard**) i opisz je – patrz rysunek 7.29.1;
- Kliknij dwukrotnie przycisk o nazwie „Chemia” i w wygenerowanym zdarzeniu wpisz kod:

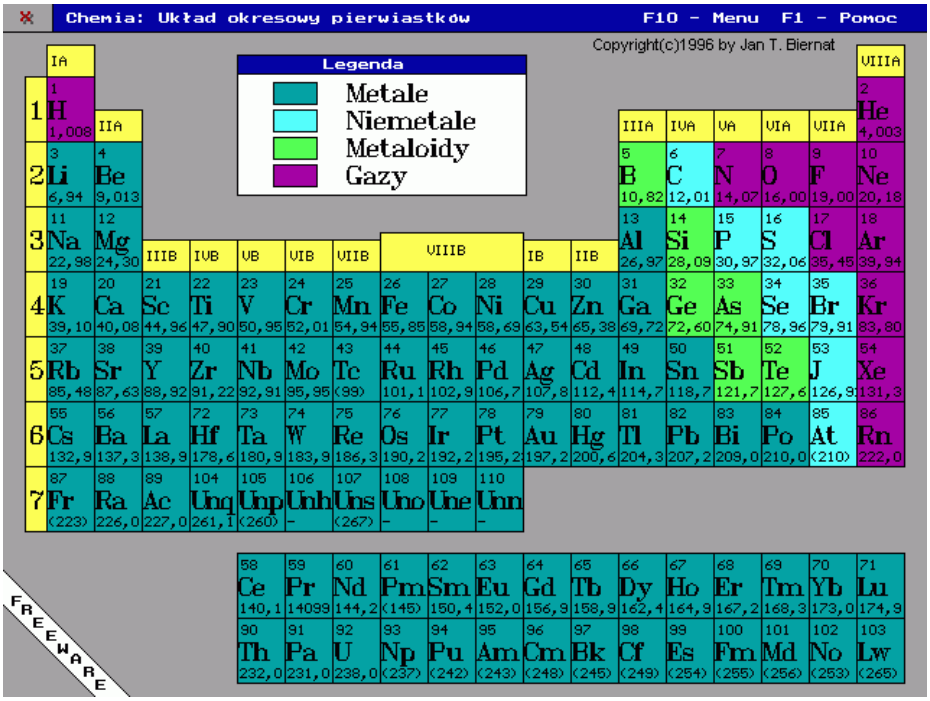
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Uruchomienie programu Chemia
  ShellExecute(Handle, 'open', PChar('chemia.exe'), '', '', sw_Normal);

  // Konwersja do typu "PChar(Lancuch)" zwraca wskaźnik do pierwszego
  // znaku łańcucha.
end;
```

- Kliknij dwukrotnie na przycisk z napisem „Kalkulator” i w wygenerowanym zdarzeniu wpisz kod:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  // Uruchomienie programu Kalkulator (calc.exe)
  ShellExecute(Handle, 'open', 'calc.exe', '', '', sw_Normal);
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

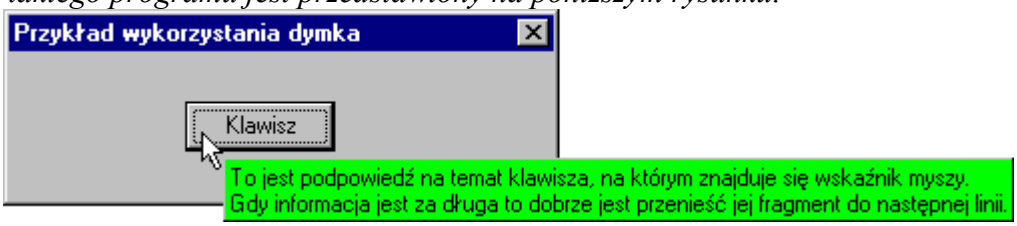
Naciśnięcie przycisku z napisem „Chemia” spowoduje uruchomienie programu CHEMIA – rysunek 7.29.2



Rysunek 7.29.2. Widok uruchomionego programu CHEMIA

Ćwiczenie 7.30. Dymki (Podpowiedzi)

Napisz program, który wyświetli opis np. przycisku po najechaniu na niego kursorem myszy. Wygląd takiego programu jest przedstawiony na poniższym rysunku.



Rysunek 7.30.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\DYMKI.

Opis:
Dymki (podpowiedzi) są komunikatem dla użytkownika mówiącym o przeznaczeniu danej funkcji programu. Komunikaty te są wyświetlane dopiero po najechaniu myszką na dany element i znajdują się w małym okienku. Gdy informacja jest za długa, to dobrze jest przenieść jej fragment do następnej linii. Dokonujemy tego przez wstawienie do naszego łańcucha w stosownym miejscu kodu Enter'a (13), aby nasza podpowiedź wyświetlona została w dwóch wierszach.
Np.

```
Button1.Hint:= 'To jest podpowiedź na temat klawisza, '+CHR(13)+
               'na którym znajduje się wskaźnik myszy.'+CHR(13)+
               'Gdy informacja jest za długa to dobrze '+
               'jest przenieść jej fragment do następnej linii';
```

Sposób wykonania:

- Wybierz klawisz **Button** z palety komponentów (karta **Standard**);
- Kliknij dwukrotnie na formie i w wygenerowanym zdarzeniu **OnCreate** wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // Ustawienie parametrów dla dymków.

  Application.HintColor:= clLime; // Ustawienie koloru dla dymku.

  // Ustawienie czasu, po którym ukaże nam się dymek z podpowiedzią.
  Application.HintPause:= 100;

  // Ustawienie czasu, określającego jak długo dymek będzie wyświetlany.
  Application.HintHidePause:= 5000;

  Button1.ShowHint:= TRUE; // Włączenie podpowiedzi.

  // Button1.Hint – Wpisanie podpowiedzi pod konkretny komponent (np. klawisz)
  Button1.Hint:= 'To jest podpowiedź na temat klawisza, na którym znajduje się wskaźnik
myszy.'+CHR(13)+ 'Gdy informacja jest za długa to dobrze jest przenieść jej fragment do
następnej linii.';
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

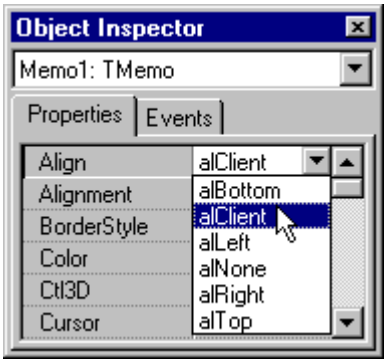
Ćwiczenie 7.31. Wczytanie pliku przez podanie jego nazwy jako parametru

Wykonaj program, który wczyta plik tekstowy w momencie uruchomienia programu.

Przedstawiony przykład znajduje się w katalogu D7CW \PARAMSTR.

Opis:
Możliwość wczytywania pliku do programu przez podanie jego nazwy jako parametru jest wygodnym rozwiązaniem zwalniającym użytkownika od wywoływania funkcji do wczytania pliku. Zamiast tego może on uruchomić program z podaniem nazwy pliku, co spowoduje uruchomienie aplikacji z automatycznym wczytaniem pliku podanego jako parametru. Umożliwia to również skojarzenie plików z konkretną aplikacją, przez co wybór danego pliku (np. w Eksploratorze) spowoduje uruchomienie się konkretnej aplikacji.

Sposób wykonania:
- Wybierz komponent **Memo** z palety komponentów (karta **Standard**);
- Kliknij na nim, w celu zaznaczenia go;
- Naciśnij klawisz funkcyjny **F11**, aby przejść do okna **Object Inspector** (Inspektora obiektów);
- Będąc na zakładce **Properties** (Właściwości) wybierz właściwość **Align** i ustaw ją na **alClient** – rysunek 7.31.1;



Rysunek 7.31.1. Widok rozwiniętej listy dostępnych ustawień dla właściwości Align

- Będąc na zakładce **Events** (Zdarzenia) wybierz właściwość **OnShow**;
- Kliknij dwukrotnie z prawej strony napisu **OnShow** (druga kolumna), w celu wygenerowania procedury;
- W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.FormShow(Sender: TObject);
begin
  // Wczytanie rysunku o nazwie podanej,
  // w trakcie uruchomienia programu.

  {
   Warunek "(ParamStr(1)<>'')" sprawdza, czy została podana
   nazwa pliku w trakcie uruchamiania programu.

   Jeżeli nazwa pliku była podana w trakcie wywoływania
   programu, to wykonaj instrukcje po słowie THEN.
```

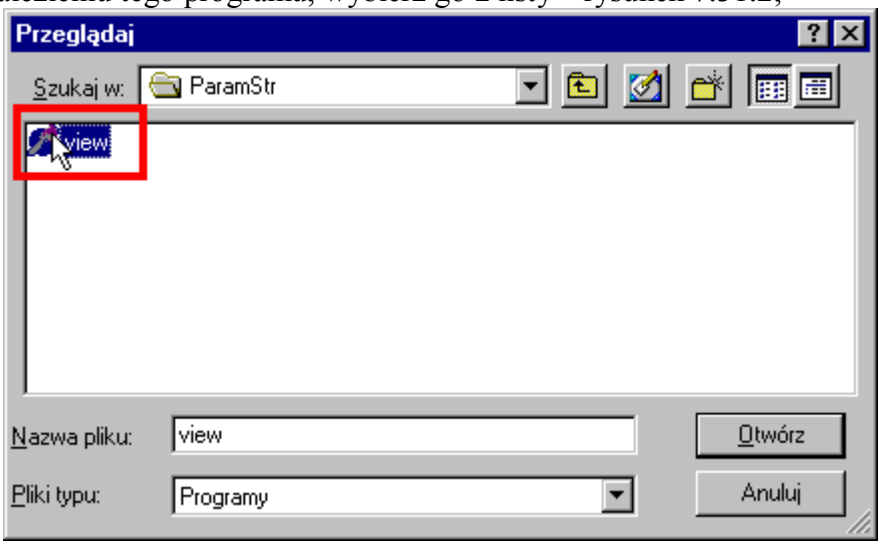
```
    W innym przypadku wykonaj instrukcje po słowie ELSE.
  }
  if (ParamStr(1)<>'') then
  begin
    // Jeżeli warunek został spełniony, to wykonaj poniższe instrukcje

    Memo1.Lines.Clear; // Wyczyszczenie komponentu "Memo1"

    // Wczytanie pliku o nazwie podanej w parametrze
    Memo1.Lines.LoadFromFile(ParamStr(1));
  end
  else
  begin
    // Zamknij program, gdy nie podano żadnej
    // nazwy w trakcie wywoływania programu
    Application.Terminate;
  end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Uwaga: Po uruchomieniu programu, zostanie on natychmiast zamknięty. Ponieważ nie wywołałeś go z parametrem.

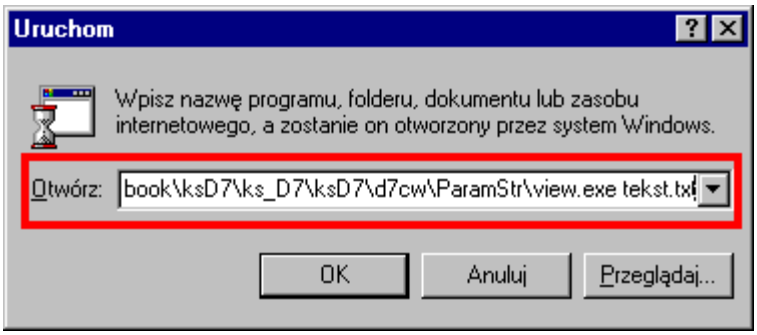
W celu uruchomienia programu z parametrem, wykonaj następujące kroki:

- Wybierz menu **START/URUCHOM**;
- Będąc w oknie **Uruchom** kliknij na klawisz **Przeglądaj...**;
- Będąc w oknie **Przeglądaj** skorzystaj z listy rozwijanej **Szukaj w**, w celu odnalezienia katalogu (w którym znajduje się program);
- Po odnalezieniu tego programu, wybierz go z listy – rysunek 7.31.2;



Rysunek 7.31.2. Widok wybranego programu

- Kliknij na klawisz **Otwórz**, co spowoduje wstawienie nazwy programu do pola edycyjnego **Otwórz** w oknie **Uruchom**;
- Dopisz nazwę drugiego pliku (tekstowego), który będzie traktowany jako parametr – rysunek 7.31.3;



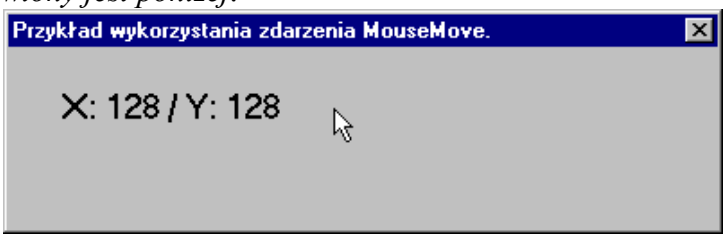
Rysunek 7.31.3. Widok wprowadzonej nazwy programu i nazwy pliku tekstowego, który traktowany jest jako parametr

- Kliknij na klawisz **OK**, w celu uruchomienia programu;

Zwróć uwagę na fakt, że drugi plik (tekstowy) traktowany jako parametr został automatycznie wczytany w momencie uruchomienia programu.

Ćwiczenie 7.32. Pozycja kursora myszy

Napisz program, który będzie pokazywał pozycję kursora myszy. Przykładowy rysunek programu przedstawiony jest poniżej.



Rysunek 7.32.1. Widok programu

Przykład znajduje się w katalogu D7CW\PKURSORA.

Sposób wykonania:

- Wybierz komponent **Label** z palety komponentów (karta **Standard**);
- Kliknij na formatce, w celu wyświetlenia jej właściwości w oknie **Object Inspector** (Inspektor Obiektów);
- Naciśnij klawisz funkcyjny **F11**, w celu przejścia do okna **Object Inspector** (Inspektor Obiektów);
- Wybierz zakładkę **Events** (Zdarzenia);
- Wybierz zdarzenie **OnMouseMove**;
- Kliknij dwukrotnie z prawej strony napisu **OnMouseMove** (druga kolumna), co spowoduje wygenerowanie procedury;
- W wygenerowanej procedurze wpisz kod:

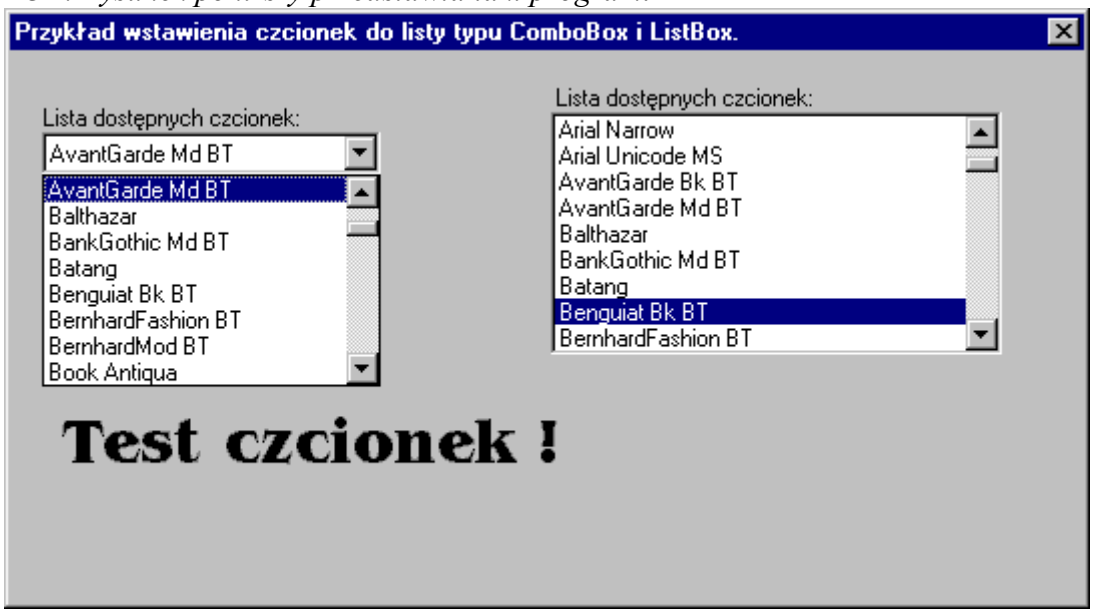
```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
var
  // Zadeklarowanie obiektu reprezentującego punkty na ekranie.
  Position: TPoint;
begin
  // Podanie pozycji kursora myszy w momencie poruszania nim.
  GetCursorPos(Position);
```

```
  // Wyświetlenie pozycji kursora myszy na ekranie za pomocą komponentu Label.
  Label1.Caption:= 'X:'+CHR(32)+IntToStr(Position.X-Left)+CHR(32)+
                '/'+CHR(32)+'Y:'+CHR(32)+IntToStr(Position.Y-Top);
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.33. Wczytanie czcionek do listy ComboBox i ListBox

Wykonaj program, który wczyta dostępne w systemie czcionki do listy COMBOBOX i LISTBOX. Rysunek poniższy przedstawia taki program.



Rysunek 7.33.1. Widok programu

Przykład znajduje się w katalogu D7CW\FONTY.

Sposób wykonania:

- Wybierz komponent **ComboBox** z palety komponentów (karta **Standard**);
- Wybierz komponent **ListBox** (karta **Standard**);
- Wybierz trzy komponenty **Label** (karta **Standard**) i ustaw je następująco:
  - Jeden nad komponentem **ComboBox**;
  - Drugi nad komponentem **ListBox**;
  - Opisz je jako „Lista dostępnych czcionek”;
  - Trzeci umieść pod komponentem **ComboBox**.
- Kliknij dwukrotnie na formie i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //-- ComboBox --
  // Wstawienie czcionek do komponentu ComboBox.
  ComboBox1.Items.Clear; // Wyczyszczenie listy.
```

```
  // Wstawienie dostępnych czcionek do listy.
  ComboBox1.Items.Assign(Screen.Fonts);

  // Wyczyszczenie zawartości edytora.
  ComboBox1.Text:= '';

  //-- ListBox --
  // Wstawienie czcionek do komponentu ListBox.
  ListBox1.Items.Clear; // Wyczyszczenie listy.

  // Wstawienie dostępnych czcionek do listy.
  ListBox1.Items.Assign(Screen.Fonts);
end;
```

- Kliknij dwukrotnie komponent **ComboBox** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  // Pokaż czcionkę po jej wybraniu.
  Label3.Font.Name:= ComboBox1.Text;
end;
```

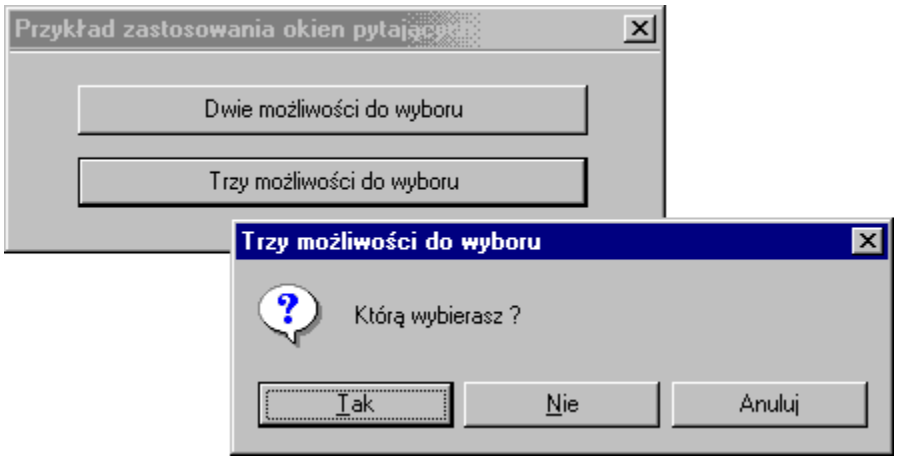
- Kliknij dwukrotnie komponent **ListBox** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
  // Pokaż czcionkę po jej wybraniu.
  if (ListBox1.ItemIndex >-1) then
  {
   Sprawdzenie czy został zaznaczony
   element, jeśli tak to wykonaj warunek „ListBox1.ItemIndex >-1”.
  }
  begin
    Label3.Font.Name:= ListBox1.Items[ListBox1.ItemIndex]; // Zmiana czcionki
  end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.34. Okna służące do zadawania pytań

Napisz program, który będzie prezentował okna służące do zadawania pytań. Rysunek poniżej przedstawia taki program.

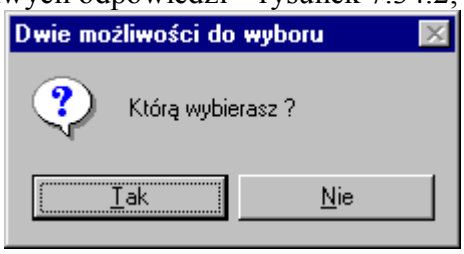


Rysunek 7.34.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\OKNAPYT.

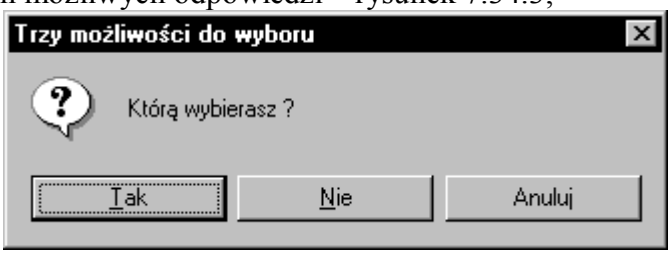
Opis okien służących do zadawania pytań:
Okna pytające służą do zadawania pytań w trakcie działania aplikacji. Rozróżnia się następujące rodzaje okien, które dają możliwość wybrania:

- Jednej z dwóch możliwych odpowiedzi – rysunek 7.34.2;



Rysunek 7.34.2. Widok okna z dwoma przyciskami

- Jednej z trzech możliwych odpowiedzi – rysunek 7.34.3;



Rysunek 7.34.3. Widok okna z trzema przyciskami

Sposób wykonania:

- Wybierz dwa komponenty **Button** z palety komponentów (karta **Standard**) i opisz je – patrz rysunek 7.34.1;
- Kliknij dwukrotnie na przycisk z napisem „Dwie możliwości do wyboru” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  numBtn: Integer; // Deklaracja zmiennej
begin
  // Dwie możliwości do wyboru
  numBtn:= 0;
  numBtn:= Application.MessageBox('Którą wybierasz ?', 'Dwie możliwości do wyboru',
```

```
        MB_ICONQUESTION or MB_YESNO);

  if (numBtn = IDYES) then
  begin
    ShowMessage('To jest PIERWSZA możliwość');
  end;

  if (numBtn = IDNO) then
  begin
    ShowMessage('To jest DRUGA możliwość');
  end;
end;
```

- Kliknij dwukrotnie na przycisk z napisem „Trzy możliwości do wyboru” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  numBtn: Integer; // Deklaracja zmiennej
begin
  // Trzy możliwości do wyboru
  numBtn:= 0;
  numBtn:= Application.MessageBox('Którą wybierasz ?', 'Trzy możliwości do wyboru',
        MB_ICONQUESTION or MB_YESNOCANCEL);

  if (numBtn = IDYES) then
  begin
    ShowMessage('To jest PIERWSZA możliwość');
  end;

  if (numBtn = IDNO) then
  begin
    ShowMessage('To jest DRUGA możliwość');
  end;

  if (numBtn = IDCancel) then
  begin
    ShowMessage('To jest TRZECIA możliwość');
  end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 7.35. Stałe rozmiary głównego okna programu

Napisz program, który uniemożliwi zmianę rozmiarów okna głównego aplikacji.

Sposób wykonania:

- Kliknij na formatce (w przypadku, gdy będzie zaznaczony jakiś komponent);

- Naciśnij klawisz funkcyjny **F11**, aby przejść do okna **Object Inspector** (Inspektora obiektów);
- Kliknij na zakładkę **Events** (Zdarzenia);
- Znajdź zdarzenie **OnResize**;
- Kliknij z prawej strony napisu **OnResize** (druga kolumna), co spowoduje wygenerowanie zdarzenia;
- W wygenerowanym zdarzeniu wpisz kod:

```
procedure TForm1.FormResize(Sender: TObject);
begin
  Width:= 320;
  Height:= 200;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Od tego momentu każda próba zmiany rozmiaru okna głównego programu, spowoduje powrót do rozmiarów wpisanych w zdarzeniu **OnResize** (w kodzie programu nazwa **FormResize**).

Ćwiczenie 7.36. ProcessMessages - odblokowanie programu w trakcie pracy

Napisz program, który będzie umożliwiał w takcie działania np. jakiejś pętli, minimalizowanie aplikacji do paska zadań.

Sposób wykonania:

- Wybierz komponent **Label** z palety komponentów (karta **Standard**);
- Wybierz kilka przycisków **Button** (karta **Standard**);
- Kliknij dwukrotnie na komponent **Button** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TT :Integer; // Zadeklarowanie zmiennej
begin
  for TT:= 0 to 32234 do
  begin
     {
      Przesłanie do komponentu Label1 skonwertowanej na tekst liczby.
      IntToStr() - konwertuje cyfry na cyfry traktowane jako tekst
     }
     Label1.Caption:= IntToStr(TT);

    Application.ProcessMessages;
    {
     "ProcessMessages"
     Procedura ta służy do chwilowego przerywania pracy
     programu, po to, by Windows mógł reagować na zdarzenia.
    }
  end;
end;
```

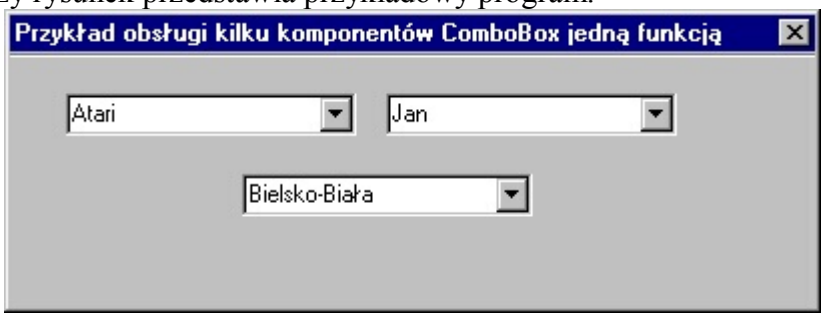
- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Pomimo tego, że program jest zajęty wykonywaniem pętli można, między innymi minimalizować aplikację do paska zadań.

8. Ćwiczenia złożone

Ćwiczenie 8.1. Obsługa kilku komponentów ComboBox jedną funkcją

Napisz program, który będzie obsługiwał kilka komponentów COMBOBOX za pomocą jednej funkcji. Poniższy rysunek przedstawia przykładowy program.



Rysunek 8.1.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\KILKA_CB.

Sposób wykonania:

- Wybierz trzy komponenty **ComboBox** z palety komponentów (karta **Standard**);
- Napisz procedurę obsługi komponentu **ComboBox**:

```
procedure TForm1.CB_WczytajListe(txtNazwaPliku :String; NazwaKomponentu
:TComboBox);
begin
  {
   Parametry procedury "CB_WczytajListe":
   txtNazwaPliku - Zmienna ta przechowuje podaną nazwę pliku
   NazwaKomponentu - Zmienna ta przechowuje podaną nazwę komponentu
  }
```

```
  // Wyczyszczenie komponentu o podanej nazwie w
  // parametrze funkcji "NazwaKomponentu"
  NazwaKomponentu.Items.Clear;



  // Wyczyść zawartość pola edycyjnego
  NazwaKomponentu.Text:= '';


  // Sprawdzenie czy plik o podanej nazwie istnieje
  if (FileExists(txtNazwaPliku) = TRUE) then
  begin
    // Jeżeli plik istnieje, to wykonaj instrukcje po słowie THEN.


    // Wczytaj zawartość pliku do listy
    NazwaKomponentu.Items.LoadFromFile(txtNazwaPliku);

    // Wstaw w polu edycyjnym pierwszy element listy
    NazwaKomponentu.Text:= NazwaKomponentu.Items[0];
  end;
end;
```

- Zadeklaruj tę procedurę w sekcji **private** (Prywatnej):

```
type
  TForm1 = class(TForm)
    Bevel1: TBevel;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    ComboBox3: TComboBox; // Deklaracja funkcji sprawdzającej
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    procedure CB_WczytajListe(txtNazwaPliku :String; NazwaKomponentu
:TComboBox);
  public
    { Public declarations }
  end;
```

- Kliknij dwukrotnie na formatce i w wygenerowanym zdarzeniu **OnCreate** (w kodzie programu zdarzenie to nosi nazwę **FormCreate**) wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // Wywołanie procedury "CB_WczytajListe" z parametrami
  CB_WczytajListe('nazw_kom.txt', ComboBox1);
  CB_WczytajListe('imiona.txt', ComboBox2);
  CB_WczytajListe('miasta.txt', ComboBox3);
```

```
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.2. Odtwarzanie ciągłe muzyki

Napisz program, który będzie odtwarzał muzykę w kółko.

Przykład znajduje się w katalogu D7CW\KOMPONENT\MEDIA_LOOP.

Sposób wykonania:

- Wybierz komponent **MediaPlayer** z palety komponentów (karta **System**);
- Wybierz komponent **OpenDialog** (karta **Dialogs**);
- Wybierz komponent **Button** (karta **Standard**) i kliknij na nim dwukrotnie, a w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Otwórz plik muzyczny
  OpenDialog1.FileName:= '';
  OpenDialog1.Execute;
  if (OpenDialog1.FileName<>'') then
  begin
     {
      Jeżeli Plik Dźwiękowy Jest Na Dysku, To
      Wykonaj Instrukcje Po Słowie THEN.
     }
     MediaPlayer1.Close; // Zamknięcie Urządzenia Odtwarzającego Dźwięk

     // Pobiera Nazwę Pliku, Który Zostanie Otwarty, a Później Odtworzony
     MediaPlayer1.FileName:= OpenDialog1.FileName;

     MediaPlayer1.Open; // Otwiera Urządzenie Odtwarzające

     {
      Ustawienie Tej Metody "MediaPlayer1.Notify" na TRUE Spowoduje
      Wywołanie Zdarzenia "OnNotify", Które Umożliwi Wychwycenie
      Momentu Zakończenia Utworu.
     }
     MediaPlayer1.Notify:= TRUE;

     MediaPlayer1.Play; // Odtwarza Wczytany Plik Muzyczny
  end;
end;
```

- Kliknij na komponent **MediaPlayer**, w celu zaznaczenia go;
- Naciśnij klawisz funkcyjny **F11**, w celu przejścia do okna **Object Inspector** (Inspektora obiektów) i wykonaj następujące kroki:
  - Kliknij na zakładkę **Events** (Zdarzenia);

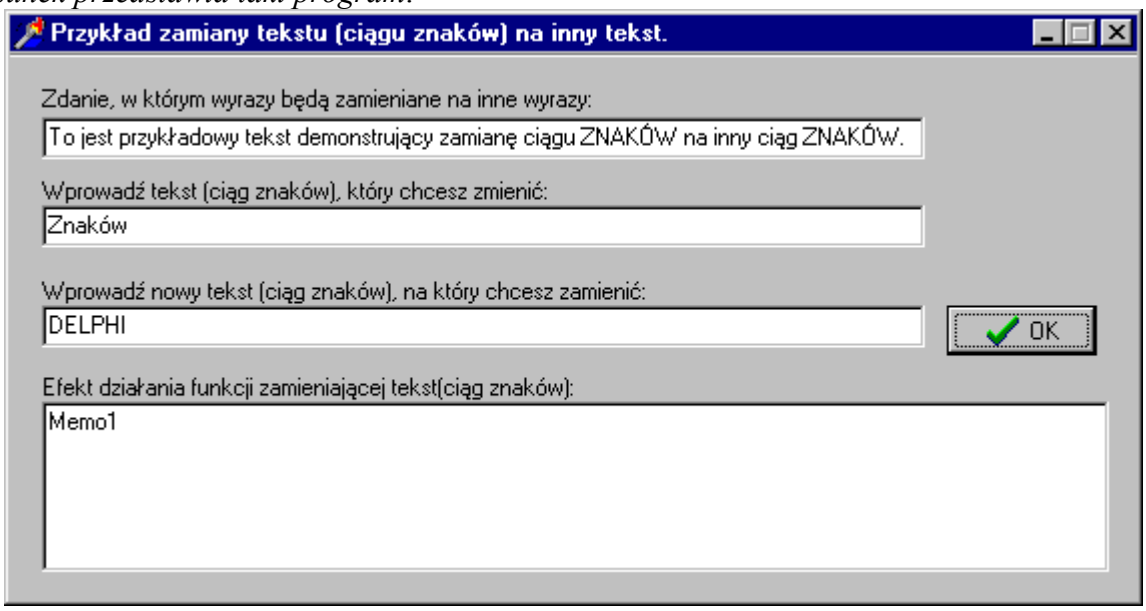
- Znajdź zdarzenie **OnNotify**;
- Kliknij z prawej strony napisu **OnNotify** (druga kolumna), co spowoduje wygenerowanie zdarzenia;
- W wygenerowanym zdarzeniu wpisz kod:

```
procedure TForm1.MediaPlayer1Notify(Sender: TObject);
begin
  // MediaPlayer1Notify = OnNotify
  {
   Jeżeli Właściwość "NotifyValue" Jest Równa
   Wartości "nvSuccessful", To Odtwórz Utwór Od Początku
  }
  if (MediaPlayer1.NotifyValue = nvSuccessful) then
    MediaPlayer1.Play
  else
    MediaPlayer1.Rewind;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.3. Zamiana znaków w tekście

Napisz program, który będzie zamieniał jeden ciąg znaków na inny ciąg znaków. Poniższy rysunek przedstawia taki program.



Rysunek 8.3.1. Widok programu

Przykład znajduje się w katalogu D7CW\ZM_ZNAK.

Sposób wykonania:

- Wybierz kilka komponentów **Label** i kilka komponentów **Edit** z palety komponentów (karta **Standard**);
- Wybierz klawisz **BitBtn** (karta **Additional**);
- Napisz funkcję zamieniającą jeden ciąg znaków na inny ciąg znaków:

```
function jbZnajdzZamien(txtText, txtFind, txtReplace: String) :String;
var
  numPos, numLen: Integer; // Deklaracja zmiennych
begin
  // Funkcja zamienia ciąg znaków na inny ciąg znaków

  numLen:= 0;
  numLen:= Length(txtFind); // Obliczenie długości ciągu znaków

  {
   AnsiUpperCase()
   Zamiana liter (ciągu znaków) na duże litery (ciągu znaków)
   ------------------------------------------------------------
   Pos(Szukany_Tekst, Tekst_w_Którym_się_Szuka)
   Zwraca pozycję znalezionego ciągu znaków
  }
  while (Pos(AnsiUpperCase(txtFind), AnsiUpperCase(txtText)) > 0) do
  begin
    {
     Pętla jest wykonywana tak długo, jak długo będzie występował
     wyszukiwany ciąg znaków. W przypadku nie znalezienia szukanego
     ciągu znaków, pętla nie wykona się ani razu.
    }

    // Podawanie pozycji znalezionego tekstu (ciągu znaków)
    numPos:= 0;
    numPos:= Pos(AnsiUpperCase(txtFind), AnsiUpperCase(txtText));

    // Usunięcie znalezionego tekstu (ciągu znaków)
    Delete(txtText, numPos, numLen);

    // Wstawienie nowego tekstu (ciągu znaków) w miejsce starego
    Insert(txtReplace, txtText, numPos);
  end;
  jbZnajdzZamien:= txtText;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „OK” i w wygenerowanej procedurze wpisz kod:

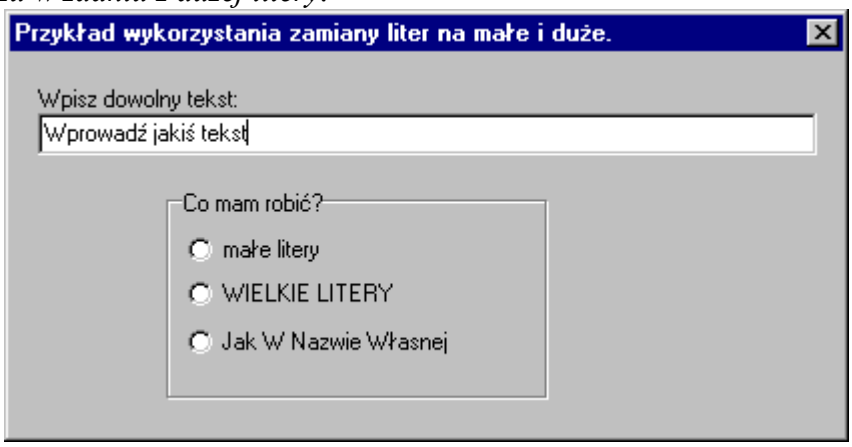
```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  // Wywołanie funkcji zamieniającej ciągi znaków.
  Memo1.Lines.Clear; // Wyczyszczenie komponentu Memo1

  // Dodanie efektu zamiany tekstu (ciągu znaków) do Memo1
  Memo1.Lines.Add(jbZnajdzZamien(
                  Edit1.Text,
                  Edit2.Text,
```

```
                Edit3.Text
                ));
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.4. Zmiana wielkości liter

Napisz program, który będzie zamieniał tekst na duże i małe litery oraz umożliwi pisanie każdego wyrazu w zdaniu z dużej litery.



Rysunek 8.4.1. Widok programu

Przykład znajduje się w katalogu D7CW\LITERY.

Sposób wykonania:

- Wybierz komponent **Label** z palety komponentów (karta **Standard**);
- Wybierz komponenty **Edit** (karta **Standard**);
- Wybierz komponent **GroupBox** (karta **Standard**);
- Wybierz trzy komponenty **RadioButton** (karta **Standard**);
- Rozmieść komponenty i opisz je – patrz rysunek 8.4.1;
- Napisz funkcję odpowiedzialną za zamianę liter (powyżej zdarzenia, np. **OnCreate** = procedurze **FormCreate**, a poniżej słowa **implementation**):

```
function ZmianaZnakow(txtString: String; chrFind: Char;
    okSwitch: Shortint): String;
{
Funkcja zmienia tekst na tekst pisany literami
małymi lub dużymi albo każdy wyraz rozpoczyna
się z wielkiej litery.
}

var // Inicjalizacja zmiennych
      TT: Integer;
  txtTemp: String;
begin
  // ZmianaZnakow
```

```
txtString:= Trim(txtString);
ZmianaZnakow:= txtString;
if (txtString<>'') then
begin
  {
   Funkcja będzie wykonana, jeżeli zmienna
   'txtString' będzie zawierała jakieś dane.
   W przeciwnym przypadku funkcja nie będzie wykonana.
  }

  {
   Zamiana tekstu na małe litery.
   Zamiana ta dokonywana jest
   przez funkcję 'AnsiLowerCase()'
  }
  if (okSwitch <= 1) then
    ZmianaZnakow:= AnsiLowerCase(txtString);

  {
   Zamiana tekstu na duże litery.
   Zamiana ta dokonywana jest
   przez funkcję 'AnsiUpperCase()'
  }
  if (okSwitch = 2) then
    ZmianaZnakow:= AnsiUpperCase(txtString);

  {
   Zamiana tekstu tak, aby wszystkie
   wyrazy rozpoczynały się dużą literą.
  }
  if (okSwitch = 3) then
  begin

    {
     Zamiana pierwszej litery w tekście na dużą,
     reszta znaków w tekście zamieniona zostaje na małe litery.
    }
    txtString:= AnsiUpperCase(Copy(txtString, 1, 1))+
              AnsiLowerCase(Copy(txtString, 2, Length(txtString)-1));


    txtTemp:= ''; // Wyczyszczenie zmiennej 'txtTemp'

    {
     Pętla wykonywana jest tyle razy ile jest
     znaków w tekście. Ilość znaków w tekście
     obliczona jest za pomocą funkcji 'Length()'
    }
    for TT:= 1 to Length(txtString) do
      if (txtString[TT-1] = CHR(32)) or
```

```
        (txtString[TT-1] = chrFind) then
        {
         Sprawdzenie czy znak na pozycji TT zmniejszonej
         o jeden, jest równy znakowi spacji lub znakowi,
         który jest wprowadzony do zmiennej 'chrFind' w
         wywołaniu funkcji. Jeżeli warunek jest spełniony
         tzn. znak leżący na pozycji TT zmniejszonej o
         jeden, będzie równy znakowi pustemu lub znakowi
         znajdującemu się w zmiennej 'chrFind', to powiększ
         znak, który leży na pozycji TT. Zmienna TT zawiera
         pozycję kolejnego znaku w tekście.


         txtTemp:= txtTemp+AnsiUpperCase(txtString[TT])
         Dodanie do zmiennej 'txtTemp' kolejnego powiększonego znaku
        }
        txtTemp:= txtTemp+AnsiUpperCase(txtString[TT])
      else
        txtTemp:= txtTemp+txtString[TT];
        {
         txtTemp:= txtTemp+ txtString[TT]
         Dodanie do zmiennej 'txtTemp' bez zmian
         kolejnego znaku.
        }
    ZmianaZnakow:= txtTemp;

   end;

  end;
end;
```

- Kliknij dwukrotnie opcję **małe litery** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  // Zmienia tekst na małe litery.
  Edit1.Text:= ZmianaZnakow(Edit1.Text, '-', 1);
end;
```

- Kliknij dwukrotnie opcję **WIELKIE LITERY** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
  // Zmienia tekst na duże litery.
  Edit1.Text:= ZmianaZnakow(Edit1.Text, '-', 2);
end;
```

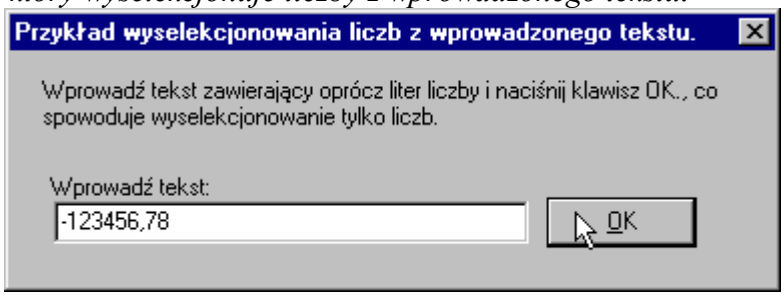
- Kliknij dwukrotnie opcję **Jak W Nazwie Własnej** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.RadioButton3Click(Sender: TObject);
begin
  {
   Zmienia tekst na tekst, w którym wszystkie
   wyrazy rozpoczynają się z dużej litery.
  }
  Edit1.Text:= ZmianaZnakow(Edit1.Text, '-', 3);
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.5. Wyselekcjonowanie liczb z tekstu

Napisz program, który wyselekcjonuje liczby z wprowadzonego tekstu.



Rysunek 8.5.1. Widok programu

Przykład znajduje się w katalogu D7CW\LICZBY.

Sposób wykonania:

- Wybierz komponent **Edit** z palety komponentów (karta **Standard**);
- Wybierz komponent **Button** (karta **Standard**);
- Napisz funkcję, która wyselekcjonuje liczby z wprowadzonego tekstu (powyżej zdarzenia, np. **OnCreate** = nazwie procedury **FormCreate**, a poniżej słowa **implementation**):

```
function jbTylkoLiczby(txtString: String; chrComma: Char): String;
{
 Funkcja wybiera z wprowadzonego tekstu
 liczby i zwraca je jako tekst.
 Jeżeli nie podamy żadnego tekstu, to
 funkcja zwróci liczbę 0 traktowaną jako tekst.
 W przypadku, gdy tekst nie zawiera liczb, to
 funkcja nie zwróci żadnej liczby
 traktowanej jako tekst.

 Przykład:
  1) wprowadzony tekst "-Przy123kła-d4owy te5k6,st 7.8"
  2) Wyselekcjonowany tekst "-123456,78"
}
var
  txtText: String; // Zadeklarowanie zmiennej tekstowej
```

```
      AA: Integer; // Zadeklarowanie zmiennej liczbowej
  begin
    // jbTylkoLiczby
    jbTylkoLiczby:= '0';
    txtString:= Trim(txtString);
    {
     Trim() - Likwiduje spacje (znaki puste) po
            obu stronach wprowadzonego tekstu
    }

    {
     Jeżeli zmienna "txtString" zawiera ciąg znaków
     to spełniony jest warunek i zostaną wykonane
     instrukcje po słowie IF...THEN
    }
    if (txtString<>'') then
    begin
      txtText:= ''; // Wyczyszczenie zmiennej tekstowej

      for AA:= 0 to Length(txtString) do
        if (txtString[AA] in ['0'..'9', chrComma]) then
          txtText:= txtText+txtString[AA];
      {
       FOR AA:= 0 TO Length(txtString) DO
       Pętla FOR...TO...DO jest wykonywana tyle
       razy ile jest znaków w wprowadzonym tekście


       Length() - Oblicza z ilu znaków
                składa się wprowadzony tekst


       IF (txtString[AA] in ['0'..'9', chrComma]) THEN
       Sprawdza czy znak jest zgodny ze znakami
       od 0 do 9 oraz znakiem reprezentującym przecinek.


       txtText:= txtText+txtString[AA];
       Dodanie do zmiennej "txtText" znaku, w
       przypadku spełnienia warunku
      }

     // Zwraca wyselekcjonowane liczby traktowane jako tekst
      jbTylkoLiczby:= Trim(txtText);

      {
       Sprawdza, czy znak minus był na początku
       wprowadzonego tekstu. Jeżeli tak, to warunek
       jest spełniony i nastąpi dodanie znaku
       minus na początku tekstu. Przykład: -ab34cd -> -34
```

```
    }
    if (txtString[1] ='-') then jbTylkoLiczby:= '-'+Trim(txtText);
  end;
end;
```

- Kliknij dwukrotnie na formatce i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  {
   Wprowadzenie przykładowego tekstu
   zawierającego również liczby
  }
  Edit1.Text:= '-Przy123kła-d4owy te5k6,st 7.8';
end;
```

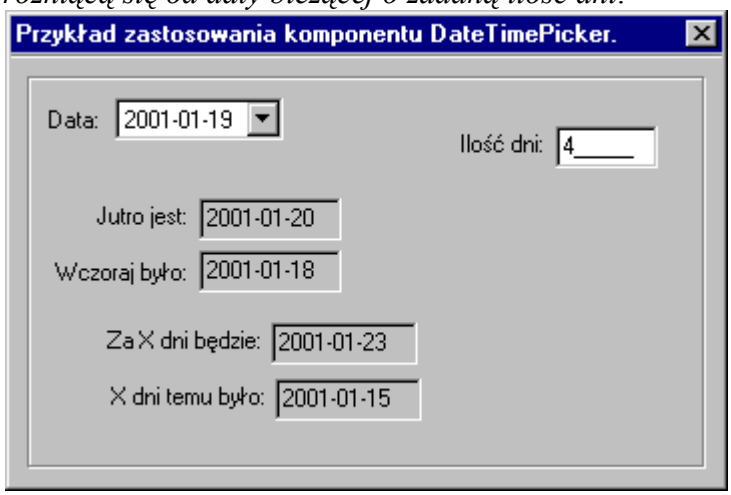
- Zaznacz klawisz i kliknij na nim dwukrotnie oraz wpisz kod w wygenerowanej procedurze:

```
procedure TForm1.bOKClick(Sender: TObject);
begin
  {
   Wywołuje funkcję do wyselekcjonowania
   liczb z wprowadzonego tekstu

   jbTylkoLiczby(Wprowadź_Tekst,
                 Znak_reprezentujący_przecinek);
  }
  Edit1.Text:= jbTylkoLiczby(Edit1.Text, ',');
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.6. Dodawanie dni do daty

Napisz program, który będzie wyświetlał datę z możliwością zmiany oraz wyświetlał datę przyszłą i przeszłą różniącą się od daty bieżącej o zadaną ilość dni.



Rysunek 8.6.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\DATA.

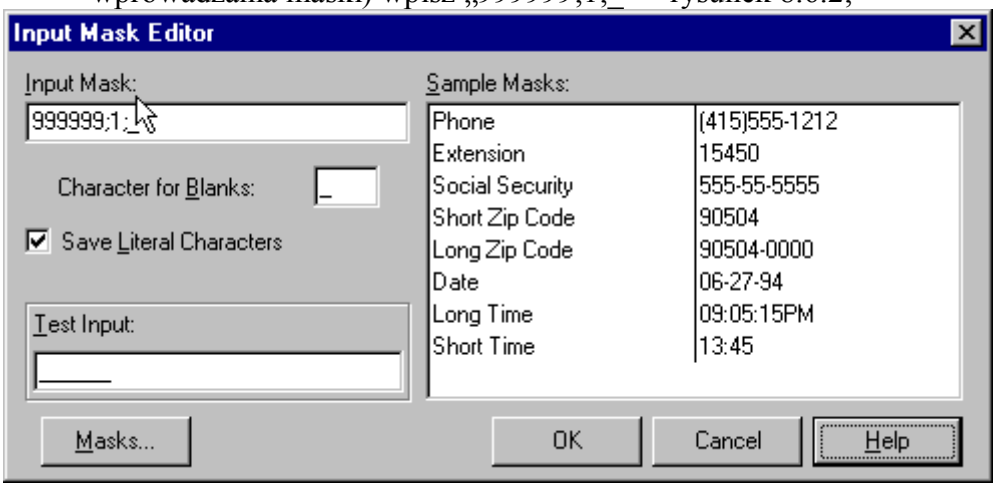
Opis komponentu:
Komponent **DateTimePicker** służy do wybierania daty za pomocą rozwijanego kalendarza oraz czasu. Komponent ten znajduje się na zakładce **Win32** palety komponentów.

UWAGA: Aby program zadziałał poprawnie, należy format daty ustawić na Rok – Miesiąc – Dzień (np. 2003-05-23).
Dokonać tego możesz wykonując następujące kroki:

- Wybierz menu **Start/Ustawienia/Panel Sterowania/Ustawienia Regionalne**;
- Będąc w oknie **Właściwości: Ustawienia regionalne** kliknij na zakładkę **Data**;
- W sekcji **Data krótka** w polu **Styl daty krótkiej** wpisz „rrrr-MM-dd”;
- Kliknij na klawisz **OK**, w celu zatwierdzenia nowych ustawień.

Sposób wykonania:

- Wybierz komponent **DateTimePicker** z palety komponentów (karta **Win32**);
- Wybierz kilka komponentów **Label** (karta **Standard**) oraz kilka komponentów **Edit** (karta **Standard**);
- Ułóż komponenty i opisz je – patrz rysunek 8.6.1;
- Właściwości wszystkich komponentów **Edit** ustaw następująco:
  - Właściwość **Color** (Kolor) ustaw na **clBtnFace** (jest to kolor szary);
  - Właściwość **ReadOnly** (Tylko odczyt) ustaw na TRUE (brak możliwości edycji);
  - Właściwość **TabStop** ustaw na FALSE (brak możliwości przechodzenia między komponentami umieszczonymi na formatce za pomocą klawisza TAB);
- Wybierz komponent **MaskEdit** (karta **Additional**);
- Zaznacz komponent **MaskEdit** i wykonaj następujące kroki:
  - Naciśnij klawisz funkcyjny **F11**, co spowoduje przejście do okna **Object Inspector** (Inspektor Obiektów);
  - Będąc na zakładce **Properties** (Właściwości) wybierz właściwość **EditMask** (Edycja maski) przez klikniecie na klawisz ...;
  - W ukazanym oknie w sekcji **Input Mask Editor** (Edytor wprowadzania maski) wpisz „999999;1;_” – rysunek 8.6.2;



Rysunek 8.6.2. Widok okna Input Mask Editor

- Napisz funkcje odpowiedzialne za dodawanie i odejmowanie dni od wprowadzonej daty (powyżej zdarzenia, np. **OnCreate** = nazwie procedury **FormCreate**, a poniżej słowa **implementation**):

```
function DodajZeroPrzed(txtZero: String): String;
begin
{
Funkcja dodaje zero do pojedynczej liczby traktowanej
jako tekst, w innym przypadku zwraca dwie liczby.

Przykład:
  1) 10 - zwróci nam liczbę dziesięć traktowaną jako tekst
  2)  3 - zwróci nam liczbę 03 traktowaną jako tekst, przez
        dodanie zera przed liczbą trzy
}
  DodajZeroPrzed:= txtZero;
  if (Length(txtZero) =1) then DodajZeroPrzed:= '0'+txtZero;
end;

function numIloscDniWMiesiacu(txtDate :String): Shortint;
// Funkcja zwraca liczbę dni w danym miesiącu
var
  numMonth, numMonthToDays: Shortint;
              numYear: Integer;
begin
  // Ilość Dni w Miesiącu

  // Wycięcie 4-ro liczbowego roku, np. 2001
  numYear:= 0;
  numYear:= StrToInt(Copy(txtDate, 1, 4));

  // Wycięcie 2-wu liczbowego miesiąca, np. 09
  numMonth:= 0;
  numMonth:= StrToInt(Copy(txtDate, 6, 2));

  // Zwraca wiadomą liczbę dni w wybranych miesiącach
  numMonthToDays:= 0;
  if (numMonth in [1, 3, 5, 7, 8, 10, 12]) then numMonthToDays:= 31;
  if (numMonth in [4, 6, 9, 11]) then numMonthToDays:= 30;
  if (numMonth = 2) then
  begin

    {
     Sprawdzenie czy podany rok jest rokiem
     przestępnym (dotyczy tylko miesiąca LUTY)
    }
    numMonthToDays:= 28;
    if ((numYear mod 4 = 0) and (numYear mod 100 <>0)) or
```

```
      (numYear mod 400 = 0) then
    begin
      numMonthToDays:= 0;
      numMonthToDays:= 29;
    end;

  end;
  numIloscDniWMiesiacu:= numMonthToDays;
end;

function txtJutroJest(txtDate: String): String;
// Funkcja zwraca datę jutrzejszą
var
  numMonth, numDay, numDayInMonth: Shortint;
                    numYear: Integer;
begin
  // Jutro Jest

  // Wycięcie 4-ro liczbowego roku, np. 2001
  numYear:= 0;
  numYear:= StrToInt(Copy(txtDate, 1, 4));

  // Wycięcie 2-wu liczbowego miesiąca, np. 09
  numMonth:= 0;
  numMonth:= StrToInt(Copy(txtDate, 6, 2));

  // Wycięcie 2-wu liczbowego dnia, np. 15
  numDay:= 0;
  numDay:= StrToInt(Copy(txtDate, 9, 2));

  {
    Przypisanie zmiennej 'numDayInMonth' ilości
    dni w danym miesiącu
  }
  numDayInMonth:= 0;
  numDayInMonth:= numIloscDniWMiesiacu(txtDate);

  // Jutro jest...
  txtJutroJest:= Copy(txtDate, 1, 5)+DodajZeroPrzed(IntToStr(numMonth))+
          '-'+DodajZeroPrzed(IntToStr(numDay+1));
  if (numDay+1 > numDayInMonth) then
  begin
    {
      Jeżeli ilość dni jest większa niż ilość dni w
      danym miesiącu to spełniony jest warunek, co
      spowoduje przesunięcie o jeden miesiąc w
      przód i dodanie 1-go dnia następnego miesiąca.
    }
    txtJutroJest:= Copy(txtDate, 1, 5)+
              DodajZeroPrzed(IntToStr(numMonth+1))+'-01';
```

```
    // Jeżeli miesiąc jest 12 to przesuń w przód o rok
    if (numMonth = 12) then txtJutroJest:= IntToStr(numYear+1)+'-01-01';
  end;
end;

function txtKiedysBedzie(txtDate: String; numDayPlus: Integer): String;
// Funkcja zwraca datę dnia za X dni
var
  txtLoopDate: String;
         TT: Integer;
begin
  // Kiedyś Będzie
  for TT:= 0 to numDayPlus-1 do
  begin
    txtLoopDate:= '';
    txtLoopDate:= txtJutroJest(txtDate); // Wywołanie funkcji txtJutroJest
    txtDate:= '';
    txtDate:= txtLoopDate;
  end;
  txtKiedysBedzie:= txtLoopDate;
end;

function txtWczorajBylo(txtDate: String): String;
// Funkcja zwraca datę wczorajszą
var
  numMonth, numDay, numDayInMonth: Shortint;
                    numYear: Integer;
begin
  // Wczoraj Było

  // Wycięcie 4-ro liczbowego roku, np. 2001
  numYear:= 0;
  numYear:= StrToInt(Copy(txtDate, 1, 4));

  // Wycięcie 2-wu liczbowego miesiąca, np. 09
  numMonth:= 0;
  numMonth:= StrToInt(Copy(txtDate, 6, 2));

  // Wycięcie 2-wu liczbowego dnia, np. 15
  numDay:= 0;
  numDay:= StrToInt(Copy(txtDate, 9, 2));

  {
    Przypisanie zmiennej 'numDayInMonth' ilości
    dni poprzedniego miesiąca
  }
  numDayInMonth:= 0;
  numDayInMonth:= numIloscDniWMiesiacu(Copy(txtDate, 1, 5)+
                        DodajZeroPrzed(IntToStr(numMonth-1))+
```

```
                    Copy(txtDate, 8, 3));

  // Wczoraj było...
  txtWczorajBylo:= Copy(txtDate, 1, 5)+DodajZeroPrzed(IntToStr(numMonth))+
        '-'+DodajZeroPrzed(IntToStr(numDay-1));
  if (numDay = 1) then
  begin
   {
    Jeżeli jest 1-szy dzień to spełniony jest warunek, co
    spowoduje cofnięcie o jeden miesiąc w tył i dodanie
    ilości dni z poprzedniego miesiąca.
   }
   txtWczorajBylo:= Copy(txtDate, 1, 5)+
          DodajZeroPrzed(IntToStr(numMonth-1))+'-'+
          DodajZeroPrzed(IntToStr(numDayInMonth));

   // Jeżeli miesiąc jest 1 to przesuń w tył o rok
   if (numMonth = 1) then txtWczorajBylo:= IntToStr(numYear-1)+'-12-31';
  end;
end;

function txtKiedysBylo(txtDate: String; numDayMinus: Integer): String;
// Funkcja zwraca datę dnia z przed X dni
var
  txtLoopDate: String;
        TT: Integer;
begin
  // Kiedyś Było
  for TT:= 0 to numDayMinus-1 do
  begin
   txtLoopDate:= '';
   txtLoopDate:= txtWczorajBylo(txtDate); // Wywołanie funkcji txtWczorajBylo
   txtDate:= '';
   txtDate:= txtLoopDate;
  end;
  txtKiedysBylo:= txtLoopDate;
end;
```

- Kliknij dwukrotnie na formatce i w wygenerowanym zdarzeniu **OnCreate** wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // Inicjacja programu
  MaskEdit1.Text:= '4'; // Wpisanie wartości 4 do komponentu MaskEdit

   // Wywołanie poszczególnych funkcji
  Edit1.Text:= txtJutroJest(DateToStr(DateTimePicker1.Date));
  Edit2.Text:= txtWczorajBylo(DateToStr(DateTimePicker1.Date));
```

```
  Edit3.Text:= txtKiedysBedzie(DateToStr(DateTimePicker1.Date),
StrToInt(Trim(MaskEdit1.Text)));

  Edit4.Text:= txtKiedysBylo(DateToStr(DateTimePicker1.Date),
StrToInt(Trim(MaskEdit1.Text)));
end;
```

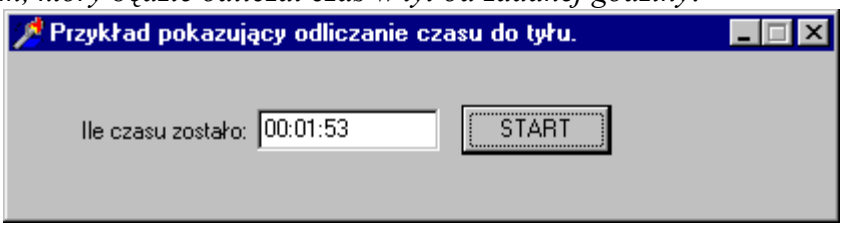
- Kliknij dwukrotnie na komponencie **DateTimePicker** i w wygenerowanym zdarzeniu **OnChange** wpisz kod:

```
procedure TForm1.DateTimePicker1Change(Sender: TObject);
begin
  // Wywołanie poszczególnych funkcji w momencie zmiany
  Edit1.Text:= txtJutroJest(DateToStr(DateTimePicker1.Date));
  Edit2.Text:= txtWczorajBylo(DateToStr(DateTimePicker1.Date));
  Edit3.Text:= txtKiedysBedzie(DateToStr(DateTimePicker1.Date),
StrToInt(Trim(MaskEdit1.Text)));
  Edit4.Text:= txtKiedysBylo(DateToStr(DateTimePicker1.Date),
StrToInt(Trim(MaskEdit1.Text)));
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.7. Odliczanie czasu w tył

Napisz program, który będzie odliczał czas w tył od zadanej godziny.



Rysunek 8.7.1. Widok programu

Przykład znajduje się w katalogu D7CW\ODLICZ_CZAS.

Sposób wykonania:

- Wybierz komponent **Label** z palety komponentów (karta **Standard**);
- Wybierz komponenty **Edit** (karta **Standard**);
- Wybierz komponent **Button** (karta **Standard**);
- Ułóż te komponenty oraz opisz – patrz rysunek 8.7.1;
- Wybierz komponent **Timer** (karta **System**) i umieść go na formatce;
- Napisz funkcje odpowiedzialne za odliczanie czasu w tył (powyżej zdarzenia, np. **OnCreate** = nazwie procedury **FormCreate**, a poniżej słowa **implementation**):

```
{
Funkcja dodaje zero do pojedynczej liczby traktowanej
jako tekst, w innym przypadku zwraca dwie liczby.
```

```
Przykład:
  1) 10 - zwróci nam liczbę dziesięć traktowaną jako tekst
  2)  3 - zwróci nam liczbę 03 traktowaną jako tekst, przez dodanie zera przed liczbą trzy
}
function FillZero(txtZero: String): String;
begin
  FillZero:= txtZero;
  if (Length(txtZero) =1) then FillZero:= '0'+txtZero;
end;

function IleCzasuPozostalo(txtCzas: String): String;
{
 Funkcja zwraca czas jaki został
 do zakończenia odliczenia.
}
var
 // Deklaracja zmiennych
  numGodz, numMinut, numSekund: Shortint;
begin
  // IleCzasuPozostalo


  numGodz:= 0; // Wyzerowanie zmiennej

  // Przypisanie zmiennej GODZIN
  numGodz:= StrToInt(Copy(txtCzas, 1, 2));


  numMinut:= 0; // Wyzerowanie zmiennej

  // Przypisanie zmiennej MINUT
  numMinut:= StrToInt(Copy(txtCzas, 4, 2));


  numSekund:= 0; // Wyzerowanie zmiennej

  // Przypisanie zmiennej SEKUND
  numSekund:= StrToInt(Copy(txtCzas, 7, 2));

  // Liczenie...
  if (numSekund = 0) then
  begin
    {
     Jeżeli zmienna 'numSekund' jest równa zeru
     to przypisz jej wartość 59, w przeciwnym
     przypadku zmniejsz wartość zmiennej
     'numSekund' o jeden.

     UWAGA:
     Tak samo jest w przypadku
```

```
    zmiennej 'numMinut' i 'numGodz'.
   }
   numSekund:= 59; // Przypisanie zmiennej wartość 59

   if (numMinut = 0) then
   begin
     numMinut:= 59; // Przypisanie zmiennej wartość 59

     if (numGodz = 0) then
       numGodz:= 23 // Przypisanie zmiennej wartość 23
     else
       numGodz:= numGodz-1; // Zmniejszanie liczby GODZ. o wartość 1

   end
   else
     numMinut:= numMinut-1; // Zmniejszanie liczby MINUT o wartość 1

  end
  else
    numSekund:= numSekund-1; // Zmniejszanie liczby SEKUND o wartość 1

  // Odliczanie...
  IleCzasuPozostalo:= FillZero(IntToStr(numGodz))+':'+
                      FillZero(IntToStr(numMinut))+':'+
                      FillZero(IntToStr(numSekund));
end;
```

- Kliknij dwukrotnie na formie i w wygenerowanym zdarzeniu **OnCreate** wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit1.Text:= '00:02:00'; // Wprowadzenie czasu

  // Wyłączenie komponentu TIMER
  Timer1.Enabled:= FALSE;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „START” i w wygenerowanym zdarzeniu **OnClick** wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Uruchomienie odliczania w tył

  Edit1.Text:= '00:02:00'; // Wprowadzenie czasu

  // Włączenie komponentu TIMER
  Timer1.Enabled:= TRUE;
end;
```

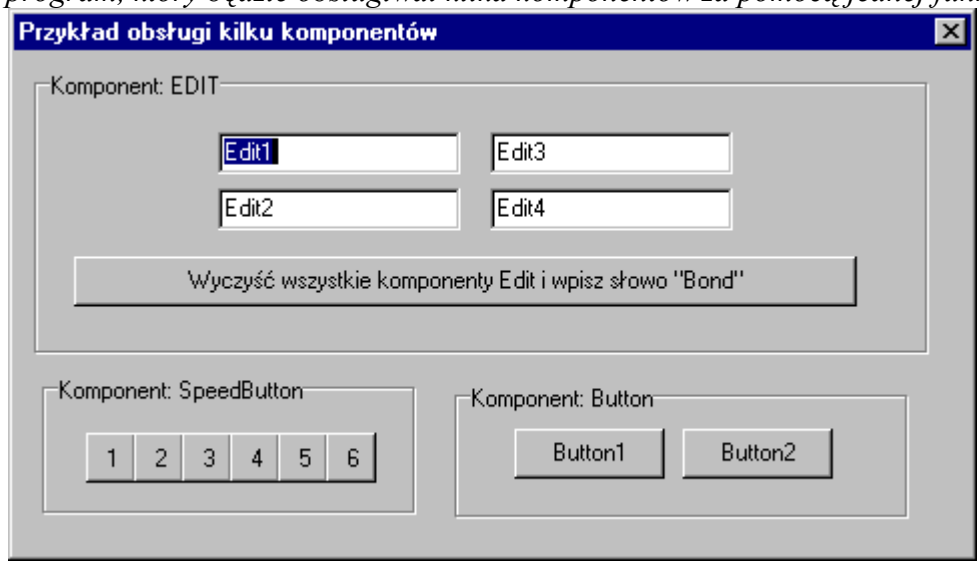
- Kliknij dwukrotnie na komponencie **Timer** i w wygenerowanym zdarzeniu **OnTimer** wpisz kod:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // Wywołanie funkcji odliczającej czas do tyłu
  Edit1.Text:= IleCzasuPozostalo(Edit1.Text);
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.8. Obsługa kilku komponentów

Wykonaj program, który będzie obsługiwał kilka komponentów za pomocą jednej funkcji.



Rysunek 8.8.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\KILKOM.

Opis:
Obsługa kilku komponentów jest bardzo dogodnym rozwiązaniem w przypadku, gdy chcemy obsłużyć np. 20 przycisków za pomocą jednego zdarzenia. Ten sposób obsługi przyczynia się do zmniejszenia kodu programu oraz do zwiększenia jego czytelności. Metodą tą możemy obsłużyć kilka komponentów jednego typu, tzn. tylko **ComboBox** lub **ListBox**. Gdyby takiej możliwości nie było, to obsługa 20 zdarzeń oddzielnie byłaby nieunikniona.

Sposób wykonania:

- Wybierz kilka komponentów **GroupBox** z palety komponentów (karta **Standard**) i opisz je – patrz rysunek 8.8.1;
- Wybierz kilka komponentów **Edit** (karta **Standard**) oraz jeden komponent **Button** (karta **Standard**) i umieść je w pierwszym **GroupBox'ie** opisanym jako „Komponent: EDIT";
- Komponent **Button** opisz jako „Wyczyść wszystkie komponenty Edit i wpisz słowo "Bond"";

- Kliknij dwukrotnie na klawisz z napisem „Wyczyść wszystkie komponenty Edit i wpisz słowo "Bond"” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bEditClick(Sender: TObject);
var
  TT: Byte; // Deklaracja zmiennej
  TC: TEdit; // Utworzenie obiektu TC
begin
  // Wyczyść wszystkie komponenty Edit i wpisz słowo "Bond"
  for TT:= 1 to 4 do // Określenie ilości komponentów na formatce (np. 4).
  begin
    // Wyszukuje komponenty o podanej nazwie.
    TC:= TEdit(FindComponent('Edit'+IntToStr(TT)));
    // Pozwala na zmianę właściwości kilku takim samym komponentom.

    TC.Font.Color:= clBlue; // Zmiana koloru na niebieski we wszystkich Edit'ach.
    TC.Text:= 'Bond'; // Wpisanie słowa "Bond" we wszystkich komponentach Edit.
  end;
end;
```

- Wybierz kilka komponentów **SpeedButton** (karta **Additional**) i umieść je w drugim **GroupBox'ie** opisanym jako „Komponent: SpeedButton”;
- Opisz je kolejno „1”, „2”, itd.;
- Kliknij dwukrotnie na pierwszy klawisz **SpeedButton** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  // Obsługa kilku przycisków za pomocą jednej funkcji.

  {
   Sender - jest referencją do obiektu, którego dotyczy dane zdarzenie
         (np. kliknięcie myszką)
   (Sender as TSpeedButton).Caption - konstrukcja ta umożliwia
      przekazanie własności, które są wspólne dla wszystkich
      komponentów SpeedButton. W tym przykładzie chodzi o
      przekazanie napisów na przyciskach.
  }

  ShowMessage('Wybrałeś:'+CHR(32)+(Sender as TSpeedButton).Caption);

  if ((Sender as TSpeedButton).Caption = '1') then ShowMessage('Jeden');
  if ((Sender as TSpeedButton).Caption = '2') then ShowMessage('Dwa');
  if ((Sender as TSpeedButton).Caption = '3') then ShowMessage('Trzy');
  if ((Sender as TSpeedButton).Caption = '4') then ShowMessage('Cztery');
  if ((Sender as TSpeedButton).Caption = '5') then ShowMessage('Pięć');
  if ((Sender as TSpeedButton).Caption = '6') then ShowMessage('Sześć');
end;
```

- Kliknij na drugi przycisk, w celu zaznaczenia go i wykonaj następujące kroki:

- Naciśnij klawisz funkcyjny **F11**, aby przyjść do okna **Object Inspector** (Inspektor obiektów);
- Będąc na zakładce **Events** (Zdarzenia) wybierz zdarzenie **OnClick**;
- Rozwiń listę zdarzeń **OnClick** (druga kolumna);
- Z rozwiniętej listy wybierz nazwę procedury obsługującej pierwszy klawisz (w naszym przykładzie jest to procedura **SpeedButton1Click**);
- Tak samo postępuj z resztą przycisków **SpeedButton**.

- Wybierz dwa klawisze **Button** (karta **Standard**) i umieść je w trzecim **GroupBox'ie** opisanym jako „Komponent: Button";
- Kliknij dwukrotnie na pierwszy klawisz **Button** i w wygenerowanej procedurze wpisz kod:

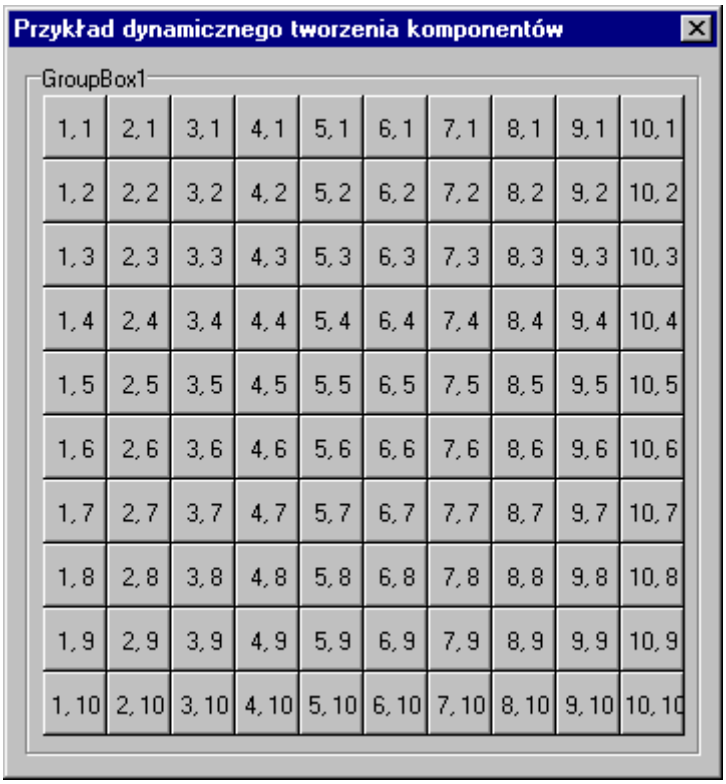
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Obsługa kilku przycisków za pomocą jednej funkcji.
  if ((Sender as TButton).Tag = 1) then ShowMessage((Sender as TButton).Caption);
  if ((Sender as TButton).Tag = 2) then ShowMessage((Sender as TButton).Caption);
  {
   Tag – jest to numer kolejnego komponentu, który ustawia się we właściwościach.
        Za pomocą tego numeru można łatwo zidentyfikować komponent.
  }
end;
```

- Kliknij drugi przycisk, w celu zaznaczenia go i wykonaj następujące kroki:
  - Naciśnij klawisz funkcyjny **F11**, aby przyjść do okna **Object Inspector** (Inspektor obiektów);
  - Będąc na zakładce **Events** (Zdarzenia) wybierz zdarzenie **OnClick**;
  - Rozwiń listę zdarzeń **OnClick** (druga kolumna);
  - Z rozwiniętej listy wybierz nazwę procedury obsługującej pierwszy klawisz (w naszym przykładzie jest to procedura **Button1Click**).

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.9. Dynamiczne tworzenie komponentów

Wykonaj program, który stworzy dynamicznie 100 klawiszy wykorzystując komponent SpeedButton oraz poukłada je w 10-ciu kolumnach po 10 wierszy.



Rysunek 8.9.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\DTWKOM.

Opis:
Możliwość dynamicznego tworzenia obiektów skraca czas pisania programu przez zastosowanie np. jednej funkcji do stworzenia kilku lub kilkunastu komponentów np. **Image**, w celu wczytania kilku lub kilkunastu obrazków. Gdyby takiej możliwości nie było, to programista musiałby na sztywno układać kilkadziesiąt komponentów, co wydłużyłoby czas pisania programu, jak również zwiększyłoby możliwość popełnienia większej ilości błędów.

Sposób wykonania:

- Wybierz komponent **GroupBox** z palety komponentów (karta **Standard**) i rozszerz go tak, aby zmieściły się kolumny i wiersze liczące po 10 klawiszy;
- Napisz procedurę obsługi zdarzenia kliknięcia o nazwie „Klik”:

```
procedure TForm1.Klik(Sender: TObject);
begin
  // Wyświetl nazwę naciśniętego klawisza
  ShowMessage((Sender as TSpeedButton).Name);
end;
```

- Zadeklaruj stałą, określającą rozmiar klawisza (pomiędzy wyrazem **uses**, a wyrazem **type**):

```
const
  {
   Zadeklarowanie Stałej Przechowującej Szerokość i Wysokości
   Klawisza do Generowania Planszy
```

```
}
numGV_RozmiarKlawisza = 32;
```

- Zadeklaruj trzy zmienne w sekcji **private** (Prywatna):
  Zmienne „numGV_PozX" i „numGV_PozY" służą do określenia pozycji klawiszy, natomiast zmienna „SB_Klawisz" służy do dynamicznego tworzenia klawiszy (w tym przykładzie wykorzystany jest komponent **SpeedButton**).

```
type
  TForm1 = class(TForm)
    GroupBox1: TGroupBox;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }

    // Zadeklarowanie Zmiennych, Które Przechowują Położenie Klawisza
    numGV_PozX, numGV_PozY :Shortint;

    // Zadeklarowanie Obiektu Typu "TSpeedButton"
    SB_Klawisz :array[1..10, 1..10] of TSpeedButton;

    procedure Klik(Sender: TObject);
  public
    { Public declarations }
  end;
```

- Kliknij dwukrotnie na formatce i w wygenerowanym zdarzeniu **OnCreate** (w kodzie programu **FormCreate**) wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  AA, BB: Shortint; // Zadeklarowanie Zmiennych
begin
  // FormCreate - Tworzenie klawiszy
  for BB:= 0 to 9 do
  begin
    for AA:= 0 to 9 do
    begin
      {
       Dynamiczne Tworzenie Klawiszy o Nazwie "SB_Klawisz",
       Których Właścicielem Jest Komponent GroupBox1
       (na którym to klawisze są tworzone)
      }
      SB_Klawisz[AA+1, BB+1]:= TSpeedButton.Create(GroupBox1);


      // Ustalenie Dodatkowych Parametrów Tworzonych Dynamicznie Klawiszy
     // Pozycja Pierwszego Klawisza (Top)
      SB_Klawisz[AA+1, BB+1].Top:= 15+(BB*numGV_RozmiarKlawisza);
```

```
        // Pozycja Pierwszego Klawisza (Left)
        SB_Klawisz[AA+1, BB+1].Left:= 9+(AA*numGV_RozmiarKlawisza);

        SB_Klawisz[AA+1, BB+1].Width:= numGV_RozmiarKlawisza; // Szerokość
        SB_Klawisz[AA+1, BB+1].Height:= numGV_RozmiarKlawisza; // Wysokość

        // Nadaj Nazwę Klawiszowi
        SB_Klawisz[AA+1, BB+1].Name:= 'k'+IntToStr(AA+1)+'_'+IntToStr(BB+1);

        // Opisz klawisz
        SB_Klawisz[AA+1, BB+1].Caption:= IntToStr(AA+1)+', '+IntToStr(BB+1);

        // Nadanie Wyglądu Wypukłego Klawiszom
        SB_Klawisz[AA+1, BB+1].Flat:= FALSE;

        // Przypisanie Do Klawisza Procedury "Klik"
        SB_Klawisz[AA+1, BB+1].OnClick:= Klik;


        // Umieszczenie Komponentów "SB_Klawisz" Tworzonych
        // Dynamicznie na Komponencie GroupBox1
        GroupBox1.InsertControl(SB_Klawisz[AA+1, BB+1]);
      end;
    end;
  end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.10. Datę w formacie Rok – Miesiąc – Dzień

Wykonaj program, który wyświetli bieżącą datę w formacie Rok – Miesiąc – Dzień, bez względu na ustawienia regionalne w systemie.

Opis:
Wyświetlanie daty w stałym formacie, np. Rok – Miesiąc – Dzień pozwala twórcy programu uwolnić się od ustawień systemowych. Dzięki temu łatwiej pisze się program.

Sposób wykonania:

- Wybierz komponent **Label** z palety komponentów (karta **Standard**);
- Napisz funkcję odpowiedzialną za dodanie zera, gdy numer miesiąca będzie jednocyfrowy (np. 5 -> 05):

```
function DodajZero(txtZero :String) :String;
begin
{
Funkcja dodaje zero do pojedynczej liczby traktowanej
jako tekst, w innym przypadku zwraca dwie liczby.

Przykład:
```

```
  1) 10 - zwróci nam liczbę dziesięć traktowaną jako tekst
  2)  3 - zwróci nam liczbę 03 traktowaną jako tekst, przez dodanie zera przed liczbą trzy
}
  DodajZero:= txtZero;
  if (Length(txtZero) =1) then DodajZero:= '0'+txtZero;
 // Jeżeli będzie spełniony warunek, to wykonaj instrukcje po słowie THEN.
// Length() – zwraca liczbę znaków. Które zawarte są w zmiennej „txtZero”
end;
```

- Napisz funkcję, która wyświetla datę w formacie Rok – Miesiąc – Dzień:

```
function DATA_DzisiajJest :String;
var
  Rok, Miesiac, Dzien :Word; // Zadeklarowanie zmiennych
begin
  // DATA_DzisiajJest
  DecodeDate(Now, Rok, Miesiac, Dzien);
  {
   DecodeDate() - Procedura ta, odpowiedzialna jest za
                 podzielenie daty podanej w
                 parametrze "Data" na rok, miesiąc i dzień.

   Funkcja "Date" - zwraca aktualną datę.

   IntToStr() - Dokonuje konwersji liczby na liczbę traktowaną jako tekst.
  Po rozdzieleniu daty do zmiennej:
            ROK – przypisywany jest rok bieżący;
           MIESIAC  - przypisywany jest miesiąc bieżący;
          DZIEN – przypisywany jest dzień bieżący.
  }
  DATA_DzisiajJest:= IntToStr(Rok)+'-'+
                 DodajZero(IntToStr(Miesiac))+'-'+
                 DodajZero(IntToStr(Dzien));
end;
```

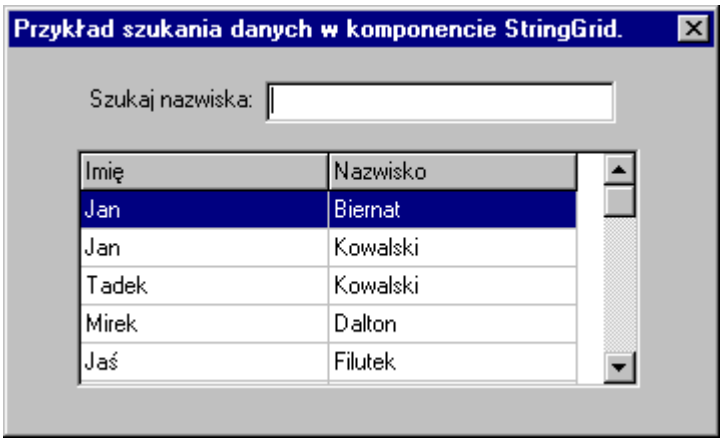
- Kliknij dwukrotnie na formatce i w wygenerowanym zdarzeniu **OnCreate** wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // FormCreate
  Label1.Caption:= DATA_DzisiajJest;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.11. Szukanie danych w komponencie StringGrid

Napisz program, który będzie umożliwiał przeszukanie rekordu z nazwiskami.



Rysunek 8.11.1 Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\STRGRID_FIND.

Sposób wykonania:

- Wybierz komponent **Label** z palety komponentów (karta **Standard**) i opisz go jako „Szukaj nazwiska:";
- Wybierz komponenty **Edit** (karta **Standard**);
- Wybierz komponent **StringGrid** (karta **Additional**);
- Napisz funkcję odpowiedzialną za wyszukanie danych (poniżej słowa **implementation** oraz poniżej dyrektywy *{$R *.DFM}*):

```
function TForm1.StringGridSearch(numCol: Integer; txtFind: String): Integer;
// Funkcja wskazuje znaleziony element
var
  TT, numLen: Integer;
    txtText: String;
begin
  // StringGridSearch
  StringGridSearch:= -1;

  {
   txtFind:= AnsiLowerCase(Trim(txtFind));
   Przypisanie wartości ze zmiennej 'txtFind' do tej samej
   zmiennej, po zlikwidowaniu zbytecznych spacji i zamianie
   wszystkich liter na małe
   ---------------------------------------------------
   AnsiLowerCase() - Zmniejszenie wszystkich liter
   Trim() - wyczyszczenie spacji po obu stronach
          wprowadzonego ciągu znaków
  }
  txtFind:= AnsiLowerCase(Trim(txtFind));


  // Obliczenie ilości znaków podanego ciągu
  numLen:= 0;
  numLen:= Length(txtFind);
```

```
{
 Przypisanie numeru kolumny, który
 podany jest w parametrze funkcji
}
StringGrid1.Col:= numCol;

// Przypisanie nr 1 pierwszemu wierszowi
StringGrid1.Row:= 1;

// Wskazanie pierwszego elementu
StringGrid1.CellRect(StringGrid1.Col, StringGrid1.Row);


// Przeszukanie wszystkich elementów
for TT:= 0 to StringGrid1.RowCount-1 do
begin

  txtText:= ''; // Wyczyszczenie zmiennej

  txtText:= AnsiLowerCase(Trim(StringGrid1.Cells[numCol, 1+TT]));
  {
  txtText:= AnsiLowerCase(Trim(StringGrid1.Cells[numCol, 1+TT]));
   Przypisanie wartości ze zmiennej 'txtFind' do tej samej
   zmiennej, po zlikwidowaniu zbytecznych spacji i zamianie
   wszystkich liter na małe
   ---------------------------------------------------
   AnsiLowerCase() - Zmniejszenie wszystkich liter
   Trim() - wyczyszczenie spacji po obu stronach
          wprowadzonego ciągu znaków
  }

  if (Copy(txtText, 1, numLen) = txtFind) then
  begin
    {
     Jeżeli element zostanie znaleziony, to zostaną
     wykonane instrukcje po konstrukcji if...then
    }

    {
     Przypisanie numeru wierszowi, w którym
     jest znaleziony element
    }
    StringGrid1.Row:= 1+TT;

    // Wskazanie znalezionego elementu
    StringGrid1.CellRect(StringGrid1.Col, StringGrid1.Row);

    // Funkcja zwraca numer wiersza,  który został wskazany
    StringGridSearch:= StringGrid1.Row;
```

```
      Break; // Przerwanie i opuszczenie pętli
    end;

  end;
end;
```

- Umieść deklarację tej funkcji (metody) w sekcji **private** (prywatnej):

```
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    StringGrid1: TStringGrid;
  private
    { Private declarations }
    function StringGridSearch(numCol: Integer; txtFind: String): Integer;
  public
    { Public declarations }
  end;
```

- Kliknij dwukrotnie na formie i w wygenerowanej procedurze **OnCreate** = **FormCreate** wpisz kod, który będzie odpowiedzialny za wypełnienie tabeli danymi:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // Wyczyszczenie zawartości komponentu Edit
  Edit1.Text:= '';

  // Wprowadzenie danych do komponentu StringGrid

  // Nagłówki
  StringGrid1.Cells[0, 0]:= 'Imię';
  StringGrid1.Cells[1, 0]:= 'Nazwisko';

  // Dane
  StringGrid1.Cells[0, 1]:= 'Jan';
  StringGrid1.Cells[1, 1]:= 'Biernat';

  StringGrid1.Cells[0, 2]:= 'Jan';
  StringGrid1.Cells[1, 2]:= 'Kowalski';

  StringGrid1.Cells[0, 3]:= 'Tadek';
  StringGrid1.Cells[1, 3]:= 'Kowalski';

  StringGrid1.Cells[0, 4]:= 'Mirek';
  StringGrid1.Cells[1, 4]:= 'Dalton';

  StringGrid1.Cells[0, 5]:= 'Jaś';
  StringGrid1.Cells[1, 5]:= 'Filutek';
```

```
  StringGrid1.Cells[0, 6]:= 'Małgosia';
  StringGrid1.Cells[1, 6]:= 'Piernik';

  StringGrid1.Cells[0, 7]:= 'Staś';
  StringGrid1.Cells[1, 7]:= 'Wiatrak';

  StringGrid1.Cells[0, 8]:= 'Nel';
  StringGrid1.Cells[1, 8]:= 'Wojna';

  StringGrid1.Cells[0, 9]:= 'Maja';
  StringGrid1.Cells[1, 9]:= 'Listek';

  StringGrid1.Cells[0, 9]:= 'Damian';
  StringGrid1.Cells[1, 9]:= 'Listek';

  StringGrid1.Cells[0, 10]:= 'Darek';
  StringGrid1.Cells[1, 10]:= 'Listek';

  StringGrid1.Cells[0, 11]:= 'Mirek';
  StringGrid1.Cells[1, 11]:= 'Listek';

  StringGrid1.Cells[0, 12]:= 'Joasia';
  StringGrid1.Cells[1, 12]:= 'Pawlik';
end;
```

- Kliknij komponent **Edit**, w celu zaznaczenia go i wykonaj następujące kroki:
  - Naciśnij klawisz funkcyjny **F11**, w celu przejścia do okna **Object Inspector** (Inspektor obiektów);
  - Będąc na zakładce **Events** (Zdarzenia) kliknij dwukrotnie obok zdarzenia **OnKeyDown** (druga kolumna), co spowoduje wygenerowanie procedury;
  - W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  // Przejście do komponentu StringGrid
  if (Key = VK_DOWN) or (Key = VK_UP) then StringGrid1.SetFocus;
end;
```

Kod ten będzie odpowiedzialny za przejście do komponentu **StringGrid**, za pomocą klawiszy strzałkowych w dół lub w górę (gdy aktywny będzie komponent **Edit**).

- Kliknij na komponent **StringGrid**, aby go zaznaczyć i wykonaj następujące kroki:
  - Naciśnij klawisz funkcyjne **F11**, w celu przejścia do okna **Object Inspector** (Inspektor obiektów);
  - Będąc na zakładce **Events** (Zdarzenia) kliknij dwukrotnie obok zdarzenia **OnKeyPress** (druga kolumna), co spowoduje wygenerowanie procedury;
  - W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.StringGrid1KeyPress(Sender: TObject; var Key: Char);
begin
  // Przejście do komponentu Edit
  Edit1.SetFocus;
end;
```

Kod ten umożliwi przejście do komponentu **Edit**, za pomocą naciśniętego dowolnego klawisza (gdy aktywny będzie komponent **StringGrid**).

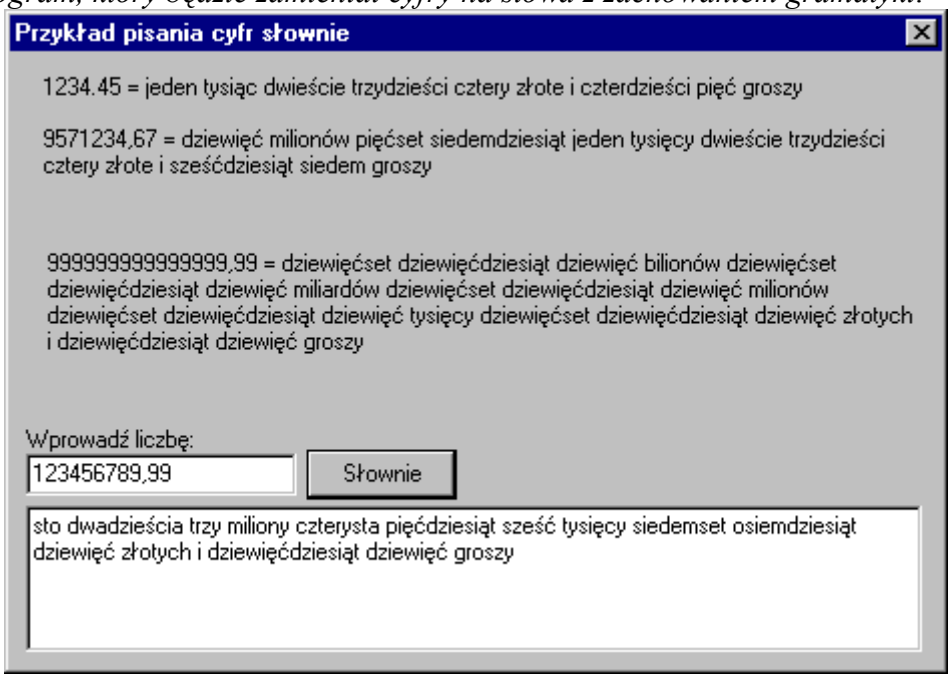
- Kliknij dwukrotnie na komponent **Edit** i w wygenerowanej procedurze wpisz kod, który jest odpowiedzialny za wywołanie funkcji wyszukującej dane:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  // Wywołanie funkcji wyszukującej dane
  StringGridSearch(1, Edit1.Text);
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.12. funkcja słownie

Napisz program, który będzie zamieniał cyfry na słowa z zachowaniem gramatyki.



Rysunek 8.12.1. Widok programu

Przykład znajduje się w katalogu D7CW\SLOWNIE.

Sposób wykonania:

- Wybierz trzy komponenty **Label** z palety komponentów (karta **Standard**);
- Wybierz komponent **Edit** z palety komponentów (karta **Standard**);

- Wybierz komponent **Memo** (karta **Standard**);
- Wybierz komponent **Button** (karta **Standard**);
- Napisz bibliotekę do konwersji liczb na słowa - przykładowa biblioteka może wyglądać tak:

```
unit Slownie;

interface

uses
 SysUtils;

function plnSlownie(txtNumber: String) :String;

implementation

{
Funkcja dodaje zero do pojedynczej liczby traktowanej
jako tekst, w innym przypadku zwraca dwie liczby.

Przykład:
  1) 10 - zwróci nam liczbę dziesięć traktowaną jako tekst
  2)  3 - zwróci nam liczbę 03 traktowaną jako tekst, przez dodanie zera przed liczbą trzy
}
function FillZero(txtZero: String): String;
begin
  FillZero:= txtZero;
  if (Length(txtZero) =1) then FillZero:= '0'+txtZero;
end;

{
Funkcja wyszukuje w łańcuchu tekstowym przecinek i zamienia go na kropkę, w
przeciwnym przypadku nie dokonuje zamiany przecinka na kropkę.

Przykład:
  1234,45 -> 1234.45
}
function CommaToDot(txtText: String): String;
var
  txtRText: String;
    numPos: Shortint;
begin
  CommaToDot:= Trim(txtText);
  if (Trim(txtText)<>'') then
  begin
    numPos:= 0;
    numPos:= Pos(',', Trim(txtText));
    if (numPos<>0) then
    begin
      txtRText:= '';
```

```
      txtRText:= Copy(Trim(txtText), 1, numPos-1)+'.'+
              Copy(Trim(txtText), numPos+1, Length(Trim(txtText)));
      CommaToDot:= Trim(txtRText);
    end;
  end;
end;

// Funkcja konwertująca liczbę na tekst
function mcIntToStr(numNumber: Longint): String;
var
  txtText: String;
begin
  txtText:= '';
  STR(numNumber, txtText); // Konwertuje wartość cyfrową na tekst.
  mcIntToStr:= txtText;
end;

// Funkcja konwertująca liczbę traktowaną jako tekst na liczbę
function mcStrToInt(txtText: String): Longint;
var
  I: Integer;
  A: Longint;
begin
  A:= 0;
  I:= 0;
  VAL(txtText, A, I); // Konwertuje tekst na wartość liczbową.
  mcStrToInt:= A;
end;

{
Funkcje zamieniające cyfry traktowane jako tekst na słowa
 np. 12356,26 -> dwanaście tysięcy trzysta pięćdziesiąt sześć złotych i
              dwadzieścia sześć groszy

}
function txtSlownieGramatyka(txtNumber: String; numPos, okPenny: Shortint) :String;
// Deklaracja tablic dla funkcji SłownieGramatyka.
const
  MM: array[1..9, 1..4] of String[20] =
    (('sto ',        'dziesięć ',        'jeden ',    'jedenaście '),
     ('dwieście ',   'dwadzieścia ',     'dwa ',      'dwanaście '),
     ('trzysta ',    'trzydzieści ',     'trzy ',     'trzynaście '),
     ('czterysta ',  'czterdzieści ',    'cztery ',   'czternaście '),
     ('pięćset ',    'pięćdziesiąt ',    'pięć ',     'piętnaście '),
     ('sześćset ',   'sześćdziesiąt ',   'sześć ',    'szesnaście '),
     ('siedemset ',  'siedemdziesiąt ',  'siedem ',   'siedemnaście '),
     ('osiemset ',   'osiemdziesiąt ',   'osiem ',    'osiemnaście '),
     ('dziewięćset ', 'dziewięćdziesiąt ', 'dziewięć ', 'dziewiętnaście '));

  NN: array[1..5, 1..3] of String[11] =
```

```
    (('złoty ',   'złote ',    'złotych '),
     ('tysiąc ',  'tysiące ',  'tysięcy '),
     ('milion ',  'miliony ',  'milionów '),
     ('miliard ', 'miliardy ', 'miliardów '),
     ('bilion ',  'biliony ',  'bilionów '));

  PP: array [1..3] of String[7] = ('grosz', 'grosze', 'groszy');

// Deklaracja zmiennych.
var
  TT, JJ, numTPos: Shortint;
        txtSay: String;
begin
  // txtSlownieGramatyka
  txtSlownieGramatyka:= '';
  if (numPos < 1) then numPos:= 1;

  if (Length(txtNumber) = 1) then txtNumber:= '00'+txtNumber;
  if (Length(txtNumber) = 2) then txtNumber:= '0'+txtNumber;

  txtSay:= '';
  if (txtNumber<>'') and (Length(txtNumber) = 3) then
  begin

    if (mcStrToInt(Copy(txtNumber, 2, 2)) in [11..19]) and
      (mcStrToInt(Copy(txtNumber, 1, 1)) = 0) then
    begin
      // 11..19 and = 0
      txtSlownieGramatyka:= MM[mcStrToInt(Copy(txtNumber, 2, 2))-10, 4]+NN[numPos, 3];
      if (okPenny >0) then
        txtSlownieGramatyka:= MM[mcStrToInt(Copy(txtNumber, 2, 2))-10, 4]+PP[3];
    end
    else
    if (mcStrToInt(Copy(txtNumber, 2, 2)) in [11..19]) and
      (mcStrToInt(Copy(txtNumber, 1, 1)) > 0) then
    begin
      // 11..19 and > 0
      txtSay:= '';
      txtSay:= MM[mcStrToInt(Copy(txtNumber, 2, 2))-10, 4]+NN[numPos, 3];
      if (okPenny >0) then
        txtSay:= MM[mcStrToInt(Copy(txtNumber, 2, 2))-10, 4]+PP[3];
      txtSlownieGramatyka:= MM[mcStrToInt(Copy(txtNumber, 1, 1)), 1]+txtSay;
    end
    else
    begin
      txtSay:= '';
      for TT:= 1 to Length(txtNumber) do
        for JJ:= 1 to 9 do
          if (Copy(txtNumber, TT, 1) = mcIntToStr(JJ)) then
            txtSay:= txtSay+MM[JJ, TT];
```

```
      numTPos:= 0;
      numTPos:= 1;
      if (mcStrToInt(Copy(txtNumber, 3, 1)) in [2..4]) then numTPos:= 2;
      if (mcStrToInt(Copy(txtNumber, 3, 1)) in [5..9]) or
        (mcStrToInt(Copy(txtNumber, 2, 1)) in [2..9]) and
        (mcStrToInt(Copy(txtNumber, 3, 1)) =1) or
        (mcStrToInt(Copy(txtNumber, 1, 1)) in [1..9]) and
        (mcStrToInt(Copy(txtNumber, 2, 1)) = 0) and
        (mcStrToInt(Copy(txtNumber, 3, 1)) = 1) or
        (mcStrToInt(Copy(txtNumber, 2, 1)) in [1..9]) and
        (mcStrToInt(Copy(txtNumber, 3, 1)) = 0) or
        (mcStrToInt(Copy(txtNumber, 1, 1)) in [1..9]) and
        (Copy(txtNumber, 2, 2) = '00') then
       begin
        numTPos:= 0;
        numTPos:= 3;
       end;
      txtSlownieGramatyka:= txtSay+NN[numPos, numTPos];
      if (okPenny >0) then txtSlownieGramatyka:= txtSay+PP[numTPos];
      if (Copy(txtNumber, 1, 3) = '000') then txtSlownieGramatyka:= '';
    end;
  end;
end;

// Rozdzielenie złotych od groszy
function txtSlowniePisz(txtNumber, txtPenny: String) :String;
var
  txtSay, txtSPenny: String;
              TT: Shortint;
begin
  // txtSlowniePisz
  txtSlowniePisz:= '';

  txtSPenny:= '';
  txtSPenny:= 'i zero groszy';
  if (mcStrToInt(txtPenny) >0) then
  begin
    txtSPenny:= '';
    txtSPenny:= 'i '+txtSlownieGramatyka(FillZero(txtPenny), 1, 1);
  end;

  // Uzupełnij zerami
  for TT:= 1 to 15-Length(txtNumber) do
    txtNumber:= '0'+txtNumber;

  txtSay:= '';
  txtSay:= 'zero złotych'+CHR(32);
  if (txtNumber <>'') and (Length(txtNumber) = 15) and
     (mcStrToInt(txtNumber) >0) then
  begin
```

```
    txtSay:= '';
    txtSay:= txtSlownieGramatyka(Copy(txtNumber, 1, 3), 5, 0)+
            txtSlownieGramatyka(Copy(txtNumber, 4, 3), 4, 0)+
            txtSlownieGramatyka(Copy(txtNumber, 7, 3), 3, 0)+
            txtSlownieGramatyka(Copy(txtNumber, 10, 3), 2, 0)+
            txtSlownieGramatyka(Copy(txtNumber, 13, 3), 1, 0);
  end;
  txtSlowniePisz:= txtSay+txtSPenny;
end;

// Wyświetlenie cyfry słownie.
function plnSlownie(txtNumber: String) :String;
var
                numComma: Shortint;
  txtBeforeComma, txtAfterComma: String;
begin
  // plnSlownie
  numComma:= 0;
  numComma:= Pos('.', CommaToDot(txtNumber));
  if (numComma<>0) then
  begin
    txtAfterComma:= '';
    txtAfterComma:= FillZero(Copy(txtNumber, numComma+1, 2));
    txtBeforeComma:= '';
    txtBeforeComma:= Copy(txtNumber, 1, numComma-1);
    plnSlownie:= txtSlowniePisz(txtBeforeComma, txtAfterComma);
  end
  else
    plnSlownie:= txtSlowniePisz(txtNumber, '-1');
end;

end.
```

- Zadeklaruj bibliotekę SLOWNIE w deklaracji **uses**, np. **uses** Slownie;
- Kliknij dwukrotnie na klawisz z opisem „Słownie” i w wygenerowanej procedurze wpisz kod:

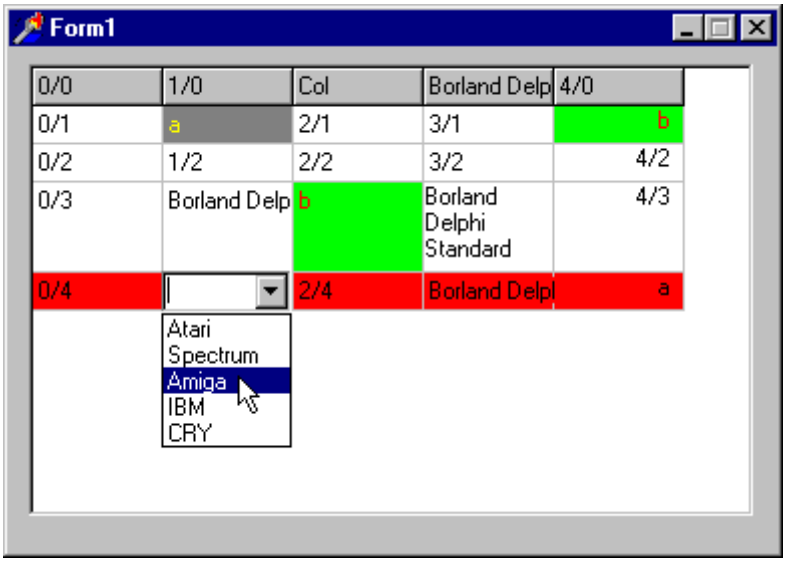
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Słownie
  Memo1.lines.Clear;
  Memo1.lines.Add(plnSlownie(eNumber.Text));
  {
    plnSlownie('Cyfra') - wywołanie funkcji, która zamienia cyfry
                          traktowane jako ciąg znaków na wyrazy.

    Przykład:
    63273,46 - sześćdziesiąt trzy tysiące dwieście siedemdziesiąt
               trzy złote i czterdzieści sześć groszy
```

```
    UWAGA:
    Funkcja bierze pod uwagę tylko dwa miejsca po przecinku.
  }
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.13. Kolorowanie komponentu StringGrid

Napisz program, który będzie prezentował następujące możliwości: - umożliwi kolorowania poszczególnych komórek oraz wybranego wiersza; - umożliwi wpisanie do komórki danych wybranych z komponentu ComboBox, który ma się ukazywać w danej komórce; - w ostatniej kolumnie napisy muszą być przesunięte do prawej krawędzi; - wybrana komórka będzie zwijała długi tekst do prawej krawędzi.



Rysunek 8.13.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\STRGRID_KOLOR.

Sposób wykonania:

- Wybierz komponent **StringGrid** (karta **Additional**) z palety komponentów;
- Wybierz komponent **ComboBox** (karta **Standard**);
- Kliknij dwukrotnie na formatce i w wygenerowanym zdarzeniu **OnCreate** (w kodzie programu **FormCreate**) wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  AA, BB: Integer; // Zadeklarowanie zmiennych
begin
  // FormCreate

  ComboBox1.Visible:= FALSE; // Schowaj Komponent ComboBox
  ComboBox1.Items.Clear; // Wyczyszczenie Listy
```

```
    // Stworzenie Listy Elementów Dla Komponentu ComboBox
    ComboBox1.Items.Add('Atari');
    ComboBox1.Items.Add('Spectrum');
    ComboBox1.Items.Add('Amiga');
    ComboBox1.Items.Add('IBM');
    ComboBox1.Items.Add('CRY');
    ComboBox1.Text:= '';



    // Wypełnienie Komórek Wartościami (Nr kolumn i Nr Wierszy)
    for AA:= 0 to StringGrid1.RowCount-1 do
      for BB:= 0 to StringGrid1.ColCount-1 do
        StringGrid1.Cells[BB, AA]:= IntToStr(BB)+'/'+IntToStr(AA);


    // Wypełnienie Wybranych Komórek Dowolną Treścią
    StringGrid1.Cells[4, 4]:= 'a';
    StringGrid1.Cells[1, 1]:= 'a';
    StringGrid1.Cells[3, 3]:= 'Borland Delphi Standard';
    StringGrid1.Cells[3, 4]:= 'Borland Delphi Standard';
    StringGrid1.Cells[3, 0]:= 'Borland Delphi Standard';
    StringGrid1.Cells[1, 3]:= 'Borland Delphi Standard';
    StringGrid1.RowHeights[3]:= 44;
    StringGrid1.Cells[4, 1]:= 'b';
    StringGrid1.Cells[2, 3]:= 'b';
    StringGrid1.Cells[2, 0]:= 'Col';
  end;
```

- Kliknij na komponent **StringGrid**, w celu zaznaczenia go i wykonania następujących czynności:
  - Naciśnij klawisz funkcyjny **F11**, w celu przejścia do okna **Object Inspector** (Inspektora obiektów);
  - Będąc na zakładce **Events** (Zdarzenia) kliknij dwukrotnie obok nazwy zdarzenia **OnDrawCell** (druga kolumna), w celu wygenerowania procedury;
  - W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.StringGrid1DrawCell(Sender: TObject; Col, Row: Integer;
  Rect: TRect; State: TGridDrawState);
var
  txtStrSG: String; // Zadeklarowanie Zmiennej
begin
  // StringGrid1DrawCell - Kolorowanie StringGrid'a
  if (Row > 0) then
  begin
     // Wykonaj instrukcje po słowie THEN, jeżeli nr wiersza jest większy od 0.

  //--- StringGridColor ---
    // Font Color
    if (Trim(StringGrid1.cells[Col, Row]) ='a') then
```

```
  begin
    // Poniższe Rozkazy(Instrukcje) Rób,
    // Jeżeli w Komórce Wpisana Jest Litera "a"

    // Wypełnij Komórkę Kolorem Szarym
    StringGrid1.Canvas.Brush.Color:= clGray;
    StringGrid1.Canvas.FillRect(Rect);

    // Określ Kolor Czcionki na Żółty
    StringGrid1.Canvas.Font.Color:= clYellow;
    StringGrid1.Canvas.TextOut(Rect.Left+2, Rect.Top+2, StringGrid1.cells[Col, Row]);

  end
  else
  if (Trim(StringGrid1.cells[Col, Row]) ='b') then
  begin
    // Poniższe Rozkazy(Instrukcje) Rób,
    // Jeżeli w Komórce Wpisana Jest Litera "b"

    // Wypełnij Komórkę Kolorem Jasno-Zielonym
    StringGrid1.Canvas.Brush.Color:= clLime;
    StringGrid1.Canvas.FillRect(Rect);

    // Określ Kolor Czcionki na Czerwony
    StringGrid1.Canvas.Font.Color:= clRed;
    StringGrid1.Canvas.Textout(Rect.Left+2, Rect.Top+2, StringGrid1.cells[Col, Row]);

  end
  else
  if (Trim(StringGrid1.cells[Col, Row]) ='') then
  begin
    // Poniższe Rozkazy(Instrukcje) Rób,
    // Jeżeli Komórka Jest Pusta

    // Wypełnij Komórkę Kolorem Białym
    StringGrid1.Canvas.Brush.Color:= clWhite;
    StringGrid1.Canvas.FillRect(Rect);

    // Określ Kolor Czcionki na Czarno
    StringGrid1.Canvas.Font.Color:= clBlack;
    StringGrid1.Canvas.Textout(Rect.Left+2, Rect.Top+2, StringGrid1.cells[Col, Row]);

  end;



//--- StringGridWordWrap ---
  if (Col = 3) and (Row = 3) then
  begin
    // Zwijaj do Lewej Napisy, Które są w Komórce o Nr Kolumny 3 i Nr Wiersza 3
```

```
    txtStrSG:= '';
    txtStrSG:= Trim(StringGrid1.Cells[Col, Row]);
    StringGrid1.Canvas.FillRect(Rect);
    DrawText(StringGrid1.Canvas.Handle,
            PChar(txtStrSG), Length(txtStrSG),
            Rect, DT_WORDBREAK or DT_LEFT);
  end;


//--- StringGridColor ---
  if (Trim(StringGrid1.cells[1, Row]) ='k') then
  begin
    // Poniższe Rozkazy(Instrukcje) Rób,
    // Jeżeli w Komórce Wpisana Jest Litera "k"

    // Wypełnij Komórkę Kolorem Zielonym
    StringGrid1.Canvas.Brush.Color:= clGreen;
    StringGrid1.Canvas.FillRect(Rect);

    // Określ Kolor Czcionki na Czarny
    StringGrid1.Canvas.Font.Color:= clBlack;
    StringGrid1.Canvas.TextOut(Rect.Left+2, Rect.Top+2, StringGrid1.cells[Col, Row]);
  end;



//--- StringGridColor ---
  if (Row = 4) then
  begin
    // Poniższe Rozkazy(Instrukcje) Rób,
    // Jeżeli w Nr Wiersza = 4

    // Wypełnij Komórkę Kolorem Czerwony
    StringGrid1.Canvas.Brush.Color:= clRED;
    StringGrid1.Canvas.FillRect(Rect);

    // Określ Kolor Czcionki na Czarny
    StringGrid1.Canvas.Font.Color:= clBlack;
    StringGrid1.Canvas.TextOut(Rect.Left+2, Rect.Top+2, StringGrid1.cells[Col, Row]);
  end;



//--- StringGridAlignment ---
  if (Col = 4) then
  begin
    // Zastosowanie Justowania do Prawej Dla Kolumny Nr 4
    txtStrSG:= '';
    txtStrSG:= Trim(StringGrid1.Cells[Col, Row]);
    StringGrid1.Canvas.FillRect(Rect);
```

```
    InflateRect(Rect, -7, 0);
    DrawText(StringGrid1.Canvas.Handle,
          PChar(txtStrSG), Length(txtStrSG),
          Rect, DT_SINGLELINE or DT_RIGHT);
  end;




 //--- Selected ---
  if (gdSelected in State) then
  begin
    // Ustaw Kolor Turkusowy Dla Kursora Wskazującego Aktualną Komórkę
    StringGrid1.Canvas.Brush.Color:= clAqua;
    StringGrid1.Canvas.FillRect(Rect);
    StringGrid1.Canvas.Font.Color:= clBlack;
    StringGrid1.Canvas.Font.Style:= StringGrid1.Canvas.Font.Style+[fsBold];
    StringGrid1.Canvas.TextOut(Rect.Left+2, Rect.Top+2, StringGrid1.Cells[Col, Row]);
  end;


 end;
end;
```

- Kliknij obok nazwy zdarzenia **OnSelectCell** (druga kolumna), w celu wygenerowania procedury;
- W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.StringGrid1SelectCell(Sender: TObject; Col, Row: Integer;
 var CanSelect: Boolean);
var
 R: TRect; // Zadeklarowanie typu TRect
 {
  TRect - służy do zapamiętania czterech punktów,
        tj. współrzędne lewego i górnego wierzchołka
        oraz współrzędne prawego i dolnego wierzchołka.
 }
begin
 // StringGrid1SelectCell
 if ((Col = 1) and (Row = 4)) or
    ((Col = 2) and (Row = 2)) then
 begin
  // Poniższe Rozkazy(Instrukcje) Rób,
  // Jeżeli w Nr Wiersza = 2 lub 4 i Nr Kolumny = 1 Lub 2


  // Pobranie Parametrów Komórki (Położenie oraz Rozmiar)
  R:= StringGrid1.CellRect(Col, Row);
  R.Left:= R.Left+StringGrid1.Left;
```

```
    R.Right:= R.Right+StringGrid1.Left;
    R.Top:= R.Top+StringGrid1.Top;
    R.Bottom:= R.Bottom+StringGrid1.Top;

    // Dopasowanie Rozmiarów Komponentu ComboBox do Wysokości i Szerokości Komórki
    ComboBox1.Left:= R.Left+1;
    ComboBox1.Top:= R.Top+1;
    ComboBox1.Width:= (R.Right+1)-R.Left;
    ComboBox1.Height:= (R.Bottom+1)-R.Top;

    // Pokaż Komponent ComboBox
    ComboBox1.Text:= '';
    ComboBox1.Visible:= TRUE;
    ComboBox1.SetFocus;
  end;
  CanSelect:= TRUE; // Umożliwienie zaznaczenia wybranej komórki
end;
```

- Kliknij na komponent **ComboBox**, w celu zaznaczenia go i wykonaj następujące kroki:
  - Naciśnij klawisz funkcyjny **F11**, w celu przejścia do okna **Object Inspector** (Inspektora obiektów);
  - Będąc na zakładce **Events** (Zdarzenia) kliknij dwukrotnie obok nazwy zdarzenia **OnClick** (druga kolumna), w celu wygenerowania procedury;
  - W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  // ComboBox1Click

  // Wstaw Wybraną Wartość z Komponentu ComboBox do Komórki
  // w Której Jest Kursor
  StringGrid1.Cells[StringGrid1.Col, StringGrid1.Row]:=
Trim(ComboBox1.Items[ComboBox1.ItemIndex]);

  ComboBox1.Visible:= FALSE; // Schowaj Komponent ComboBox
  StringGrid1.SetFocus; // Uaktywnij Komponent StringGrid
end;
```

  - Kliknij dwukrotnie obok nazwy zdarzenia **OnExit** (druga kolumna), w celu wygenerowania procedury;
  - W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.ComboBox1Exit(Sender: TObject);
begin
  // ComboBox1Exit
  ComboBox1.Visible:= FALSE; // Schowaj Komponent ComboBox
  StringGrid1.SetFocus; // Uaktywnij Komponent StringGrid
end;
```

- Kliknij dwukrotnie obok nazwy zdarzenia **OnKeyPress** (druga kolumna), w celu wygenerowania procedury;
- W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
begin
  // ComboBox1KeyPress
  if (Key = CHR(27)) then
  begin
    ComboBox1Exit(Sender); // Wywołaj Procedurę Po Naciśnięciu Klawisza ESC
  end
  else
  begin
    //TODO: write here
  end;
end;
```

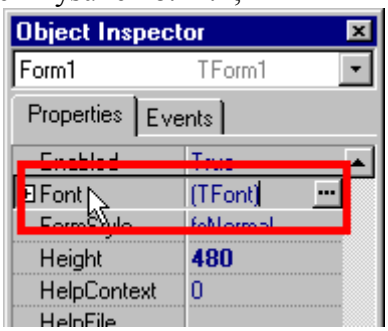
- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.14. Polskie znaki (Ogonki)

Ustaw stronę kodową polskich znaków dla formatki i sterowników baz danych Paradox.

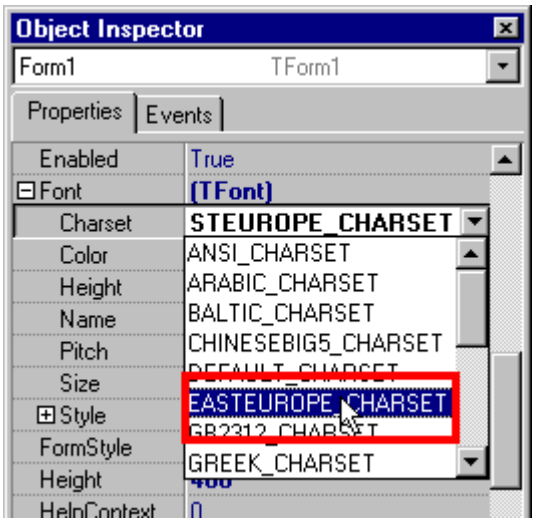
Sposób wykonania:

- Kliknij formatkę;
- Naciśnij klawisz funkcyjny **F11**, aby przejść do okna **Object Inspector** (Inspektor Obiektów);
- Będąc na zakładce **Properties** (Właściwości) rozwiń właściwość **Font** dwukrotnym kliknięciem na tej nazwie – rysunek 8.14.1;



Rysunek 8.14.1. Widok wskazanej właściwości Font

- Rozwiń listę właściwości **Charset** – rysunek 8.14.2;
- Z listy tej, wybierz stronę kodową **EASTEUROPE_CHARSET** – rysunek 8.14.2;



Rysunek 8.14.2. Widok wybranej strony kodowej EASTEUROPE_CHARSET

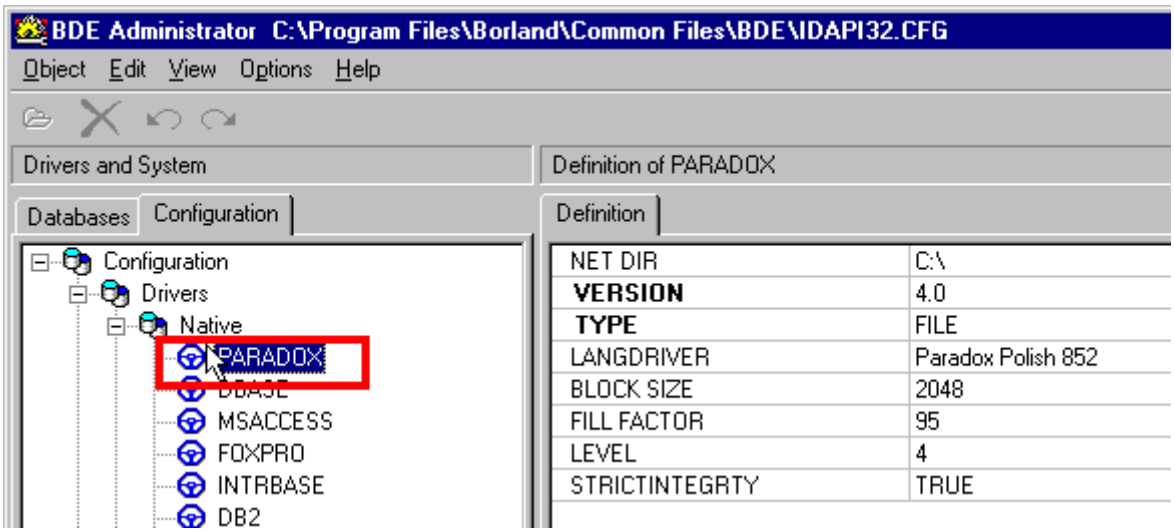
Czcionki ustawione są na stronie kodowej Europy Środkowo-Wschodniej.

W celu ustawienia strony kodowej dla sterowników baz danych typu PARADOX należy wykonać następujące kroki:

- Wybierz menu **Start/Ustawienia/Panel sterownia**;
- W oknie **Panelu sterowania** wybierz **BDE Administrator** – rysunek 8.14.3;

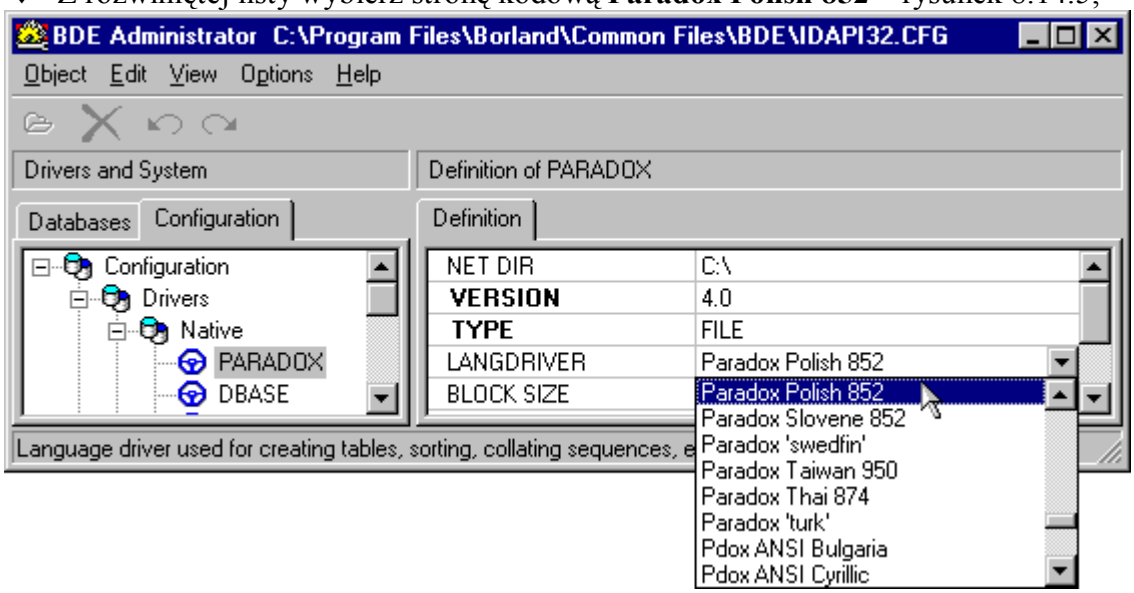
Rysunek 8.14.3. Widok wskazanej ikony BDE Administrator

- Wybierz zakładkę **Configuration** (Konfiguracja);
- Rozwiń drzewo **Configuration** (**Configuration/Drives/Native/Paradox**), aż dojdziesz do sterownika **Paradox** – rysunek 8.14.4;



Rysunek 8.14.4. Widok wybranego sterownika

- W zakładce **Definition** (Definicja), która znajduje się po prawej stronie okna **BDE Administrator'a** rozwiń listę **LANGDRIVER**;
- Z rozwiniętej listy wybierz stronę kodową **Paradox Polish 852** – rysunek 8.14.5;



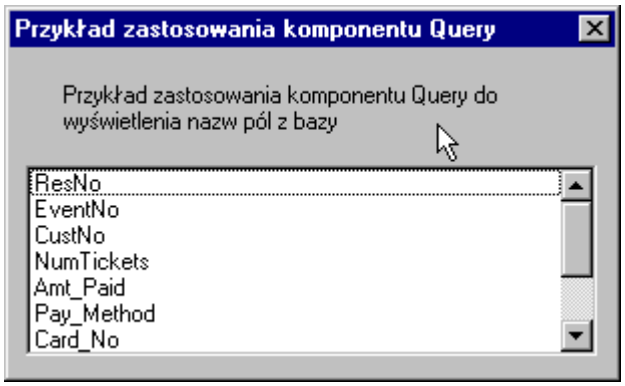
Rysunek 8.14.5. Widok wybranej strony kodowej Paradox Polish 852

- Kliknij na klawisz **Apply** (Zastosuj) , w celu zapisania nowych ustawień.

Od tego momentu baza danych typu PARADOX wykorzystuje polską stronę kodową.

Ćwiczenie 8.15. Query - Wyświetlenie nazw kolumn z bazy

Napisz program, który będzie umożliwiał odczytanie nazw kolumny występujących w bazie.



Rysunek 8.15.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\BAZY_POLA.

Opis komponentu:

Query jest komponentem służącym do reprezentowania danych, które są efektem zadanego zapytania SQL w stosunku do jednej lub większej ilości tabel.

Sposób wykonania:

- Wybierz komponent **Query** z palety komponentów (karta **Data Access**);
- Wybierz komponent **ListBox** (karta **Standard**);
- Kliknij dwukrotnie na formatce, co spowoduje wygenerowanie zdarzenia **OnCreate** oraz wpisz w wygenerowanej procedurze kod:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  TT: Integer; // Deklaracja zmiennej TT
begin
  {
   Tu wpisujemy instrukcje, które są
   wykonywane w momencie tworzenia formatki.
  }


  // Zapytanie SQL
  Query1.Close; // Zamknięcie komponentu, w celu umożliwienia wymiany instrukcji SQL
  Query1.SQL.Clear; // Czyszczenie zapytania SQL

  Query1.SQL.Add('SELECT * FROM “Reservat.db”');
  {
   SELECT * FROM “Reservat.db”
   Wyświetla zawartość tablicy RESERVAT.DB

   SELECT - Używany jest do formułowania zapytań do
          bazy danych w celu uzyskania informacji.

   * - Oznacza wyświetlenie wszystkich kolumn z danej bazy.
      Gdyby była napisana nazwa kolumny (np. name) zamiast
```

```
      gwiazdki to wyświetlona zostałaby kolumna o nazwie "Name".

    FROM - określa z jakiej tablicy lub z jakich tablic są pobierane dane.
          W naszym przypadku jest to tablica o nazwie "Country.db".
   }

   Query1.Open; // Otwarcie bazy


   //-- Odczytanie nazw pól z bazy danych --
   ListBox1.Items.Clear;
   for TT:= 0 to Query1.FieldCount-1 do
     ListBox1.Items.Add(Trim(Query1.Fields[TT].DisplayName));
   {
     ListBox1.Items.Clear;
     Wyczyszczenie zawartości komponentu ListBox

     for TT:= 0 to Query1.FieldCount-1 do
     Wykonanie pętli tyle razy ile jest pól w bazie danych

     Query1.FieldCount - Zwraca liczbę pól w bazie

     ListBox1.Items.Add() - Dodaje element do listy

     Trim()
     Likwiduje puste znaki (Spacje) po obu stronach tekstu (ciągu znaków)

     Query1.Fields[Index].DisplayName
     Wyświetla nazwę pola występującego w bazie o zadanym indeksie
   }
  end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.16. Wyświetlenie tabeli w komponencie StringGrid

Napisz program, który wyświetli zawartość bazy (Tabeli) w komponencie StringGrid i będzie umożliwiał wyszukanie stolicy państwa.

Rysunek 8.16.1. Widok programu

Przykład znajduje się w katalogu D7CW\KOMPONENT\BAZY_STRGRID.

Opis komponentu:

StringGrid (arkusz pól edycyjnych) jest komponentem, który umożliwia wyświetlenie informacji w sposób tabelaryczny lub zrobienie prostego programu kalkulacyjnego. Komponent ten znajduje się na karcie **Standard** palety komponentów.

Sposób wykonania:

- Wybierz komponent **Query** z palety komponentów (karta **Data Access**);
- Wybierz komponent **GroupBox** (karta **Standard**) i we właściwości **Caption** wpisz wyraz „Szukaj”;
- Wybierz komponent **Label** (karta **Standard**) i kliknij na komponencie **GroupBox1**, który jest opisany jako „Szukaj”;
- Zaznacz komponent **Label** i we właściwości **Caption** wpisz tekst „Wpisz stolicę”;
- Wybierz komponent **Edit** (karta **Standard**) i kliknij na komponencie **GroupBox1**, który jest opisany jako „Szukaj”;
- Wybierz komponent **GroupBox** (karta **Standard**) i we właściwości **Caption** wpisz wyraz „Lista”;
- Wybierz komponent **StringGrid** (karta **Additional**) i kliknij na komponencie **GroupBox2**, który jest opisany jako „Lista”;
- Kliknij dwukrotnie na formatce, co spowoduje wygenerowanie zdarzenia **OnCreate** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  {
   Tu wpisujemy funkcje, które są wykonywane w
   momencie tworzenia formatki.
  }
  Edit1.Text:= ''; // Wyczyszczenie komponentu Edit1
```

```
end;
```

- Naciśnij klawisz funkcyjny **F11**, w celu przejścia do okna **Object Inspector** (Inspektor obiektów);
- Będąc na zakładce **Events** (Zdarzenia) i kliknij dwukrotnie obok nazwy zdarzenia **OnShow** (druga kolumna) oraz wpisz kod, w wygenerowanej procedurze:

```
procedure TForm1.FormShow(Sender: TObject);
var
  TT: Integer; // Deklaracja zmiennej "TT"
begin
  {
   Tu wpisujemy funkcje, które są wykonywane w
   momencie otwierania formatki.
  }

  // Ustawienia dla komponentu StringGrid
  StringGrid1.Cells[0, 0]:= 'Państwo'; // Tytuł kolumny
  StringGrid1.ColWidths[0]:= 144; // Szerokość kolumny

  StringGrid1.Cells[1, 0]:= 'Stolica'; // Tytuł kolumny
  StringGrid1.ColWidths[1]:= 199; // Szerokość kolumny

  StringGrid1.RowCount:= 2; // Ilość wierszy

  // Zapytanie SQL
  Query1.Close; // Zamknięcie komponentu, w celu umożliwienia wymiany instrukcji SQL
  Query1.SQL.Clear; // Czyszczenie zapytania SQL

  Query1.SQL.Add('SELECT * FROM „Country.db” ORDER BY Name');
  {
   SELECT * FROM „Country.db” ORDER BY Name
   Wyświetla zawartość tablicy Country.db posortowaną
   alfabetycznie według kolumny "Name"

   SELECT - Używany jest do formułowania zapytań do
          bazy danych w celu uzyskania informacji.

   * - Oznacza wyświetlenie wszystkich kolumn z danej bazy.
      Gdyby była napisana nazwa kolumny (np. name) zamiast
      gwiazdki to wyświetlona zostałoby kolumna o nazwie "Name".

   FROM - określa z jakiej tablicy lub z jakich tablic są pobierane dane.
         W naszym przypadku jest to tablica o nazwie "Country.db".

   ORDER BY - klauzula ta służy do sortowania według wybranej kolumny.
            W naszym przykładzie jest to kolumna "Name".
  }

  Query1.Open; // Otwarcie bazy
```

```
  // Ustawienia na pierwszy rekord w bazie
  Query1.First;

  {
   Ustawia liczbę wierszy w komponencie StringGrid,
   według ilości rekordów w bazie
  }
  StringGrid1.RowCount:= 2+Query1.RecordCount;

  TT:= 0; // Przypisanie zmiennej "TT" wartości 0

  {
   Odczytywanie rekordów z bazy.
   Odczyt zostanie zakończony dopiero po odczytaniu
   ostatniego rekordu w bazie.
  }
  while not(Query1.EOF) do
  begin

    {
     Odczytanie Nazwy z kolumny NAME i przypisanie
    jej do pierwszej kolumny komponentu StringGrid.
    }
    StringGrid1.Cells[0, 1+TT]:= Trim(Query1.FieldByName('Name').AsString);

    {
    Odczytanie Stolicy z kolumny CAPITAL i przypisanie
   jej do drugiej kolumny komponentu StringGrid.
    }
    StringGrid1.Cells[1, 1+TT]:= Trim(Query1.FieldByName('Capital').AsString);

    Inc(TT); // Licznik wierszy

    // Przesunięcie do następnego rekordu
    Query1.Next;
  end;
end;
```

- Kliknij dwukrotnie na komponent **Edit** i w wygenerowanym zdarzeniu **OnChange** wpisz kod:

```
procedure TForm1.Edit1Change(Sender: TObject);
var
  TT, numCLS: Integer; // Zadeklarowanie zmiennych
begin
  {
   Tu wpisujemy funkcje, które będą wykonywane w
   momencie zmiany zawartości komponentu Edit.
  }
```

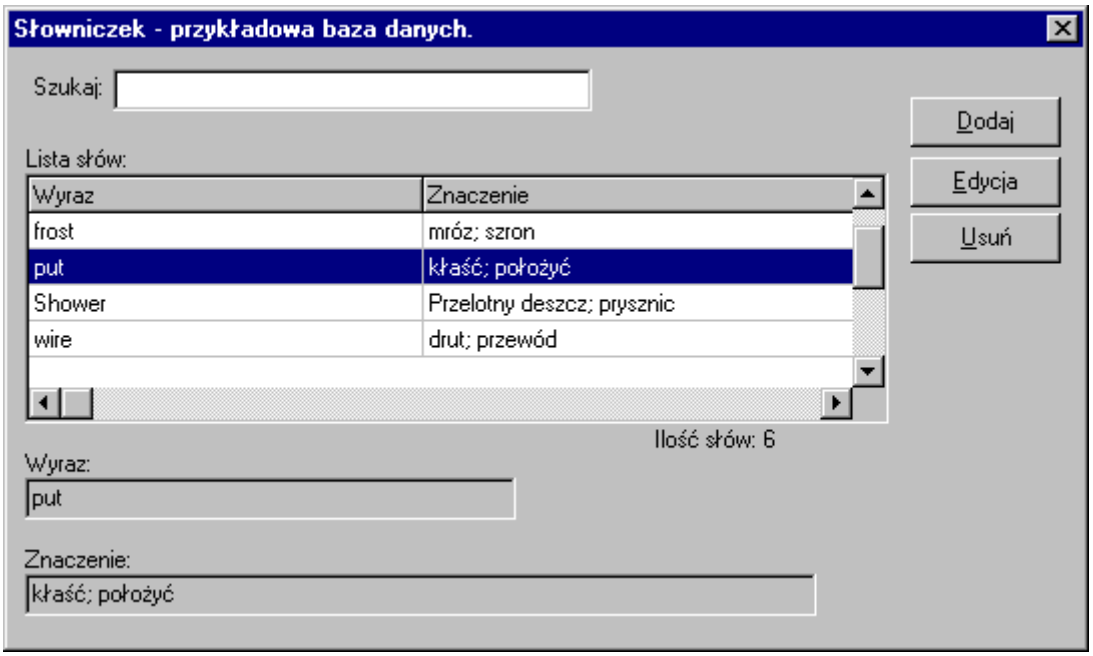
```
Query1.Close; // Zamknięcie komponentu, w celu umożliwienia wymiany instrukcji SQL
Query1.SQL.Clear; // Czyszczenie zapytania SQL

Query1.SQL.Add('SELECT * FROM „Country.db”'+
          'WHERE UPPER(Capital) LIKE :p_Capital '+
          'ORDER BY Name');
{
 WHERE - Po słowie WHERE występuje predykat, który składa
       się z jednego lub więcej wyrażeń.
       W tym przypadku UPPER(Capital) LIKE :p_Capital.

 UPPER() - Konwertuje ciąg znaków na ciąg znaków pisany dużymi literami.

 LIKE - Wyrażenie to pozwala nam na poszukiwanie określonego ciągu znaków.
}

{
 Przekazanie ciągu znaków jako parametru
 według którego nastąpi wyszukiwanie (pole Capital).
 % - Pozwala na wyświetlenie wszystkich wyrazów
    zaczynających się od litery np. A, a dzięki
    operatorowi "%" dalsze litery nie są brane pod uwagę.
}
Query1.Params[0].AsString:= UpperCase(Trim(Edit1.Text))+'%';

Query1.Open; // Otwarcie bazy

// Ustawienia na pierwszy rekord w bazie
Query1.First;

{
 Ustawia liczbę wierszy w komponencie StringGrid,
 według ilości rekordów w bazie
}
StringGrid1.RowCount:= 2+Query1.RecordCount;

TT:= 0; // Przypisanie zmiennej "TT" wartości 0

{
 Odczytywanie rekordów z bazy.
 Odczyt zostanie zakończony dopiero po odczytaniu
 ostatniego rekordu w bazie.
}
while not(Query1.EOF) do
begin

  {
   Odczytanie Nazwy z kolumny NAME i przypisanie
  jej do pierwszej kolumny komponentu StringGrid.
```

```
    }
    StringGrid1.Cells[0, 1+TT]:= Trim(Query1.FieldByName('Name').AsString);

    {
    Odczytanie Stolicy z kolumny CAPITAL i przypisanie
    jej do drugiej kolumny komponentu StringGrid.
    }
    StringGrid1.Cells[1, 1+TT]:= Trim(Query1.FieldByName('Capital').AsString);

    Inc(TT); // Licznik wierszy

    // Przesunięcie do następnego rekordu
    Query1.Next;
  end;

  // Wyczyść wiersze
  for numCLS:= TT+1 to StringGrid1.RowCount-1 do
  begin

    // Wyczyszczenie wiersza w pierwszej kolumnie
    StringGrid1.Cells[0, numCLS]:= '';

    // Wyczyszczenie wiersza w drugiej kolumnie
    StringGrid1.Cells[1, numCLS]:= '';
  end;
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.17. Wprowadzenie do baz danych

Napisz słownik, z możliwością sortowania i wyszukiwania według kolumny WYRAZ. Pisząc program wykorzystaj bazę Paradox.



Rysunek 8.17.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\KOMPONENT\BAZY_SLOWNIK.

Opis:
Baza danych jest to tabela lub kilka tabel ze sobą powiązanych. Każda tabela składa się z kolumn i wierszy. Każdy wiersz to jeden rekord na który składa się jedna kolumna lub kilka kolumn.

Wygląd przykładowej tabeli:

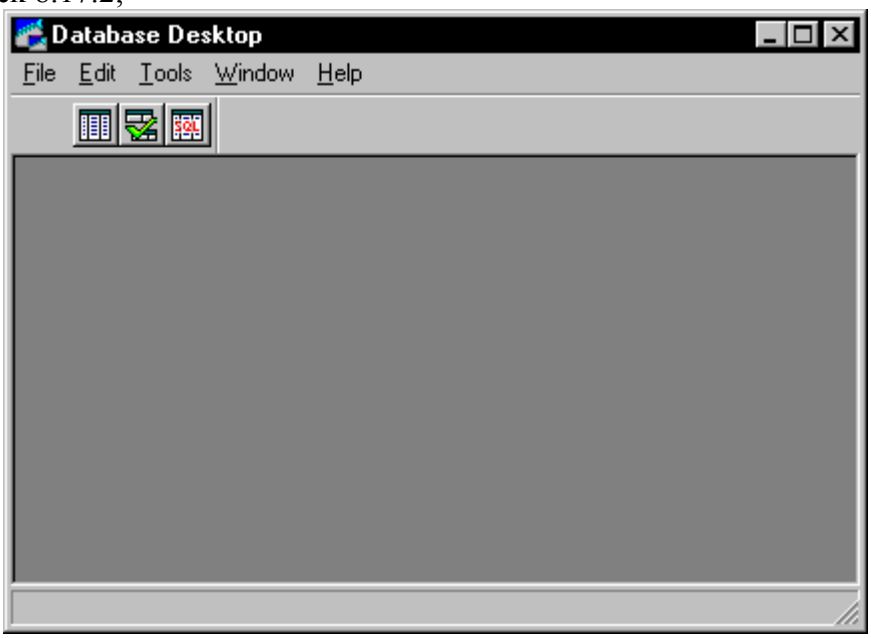
Nazwisko	Imię	Miasto
Kowalski	Tadeusz	Warszawa
Kanarek	Marian	Gdańsk
Kanapka	Andrzej	Wrocław

Sposób wykonania:

- Wybierz kilka komponentów **Label** (karta **Standard**) i opisz je – patrz rysunek 8.17.1;
- Wybierz komponent **Edit** (karta **Standard**);
- Wybierz komponent **DBGrid** (karta **Data Controls**);
- Wybierz dwa komponenty **DBEdit** (karta **Data Controls**);
- Wybierz trzy klawisze **Button** z palety komponentów (karta **Standard**) i opisz je – patrz rysunek 8.17.1;
- Wybierz następujące komponenty: - **Query** (karta **Data Access**), - **DataSource** (karta **Data Access**);

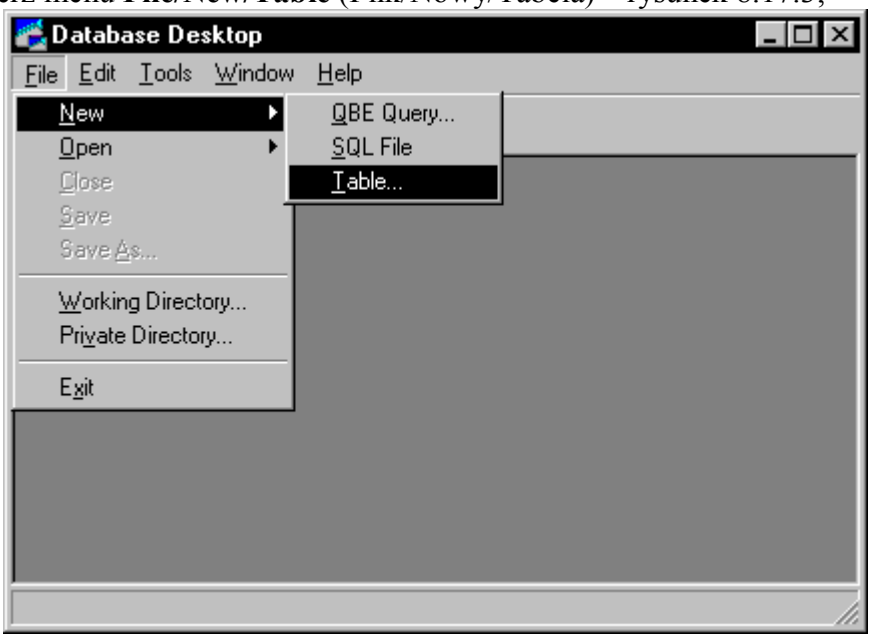
W celu utworzenia tabeli wykonaj następujące czynności:

- Po uruchomieniu Delphi wybierz menu **Tools/Database Desktop** (Narzędzia/Baza danych), co spowoduje otwarcie programu **Database Desktop** (Baza danych) – rysunek 8.17.2;



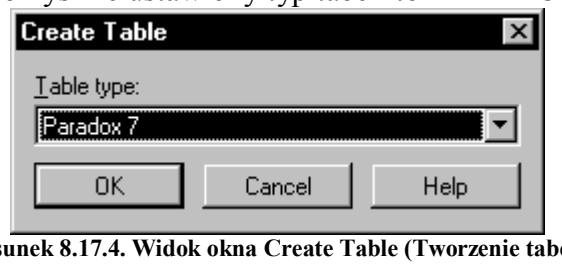
Rysunek 8.17.2. Widok uruchomionego programu Database Desktop

- Wybierz menu **File/New/Table** (Plik/Nowy/Tabela) – rysunek 8.17.3;



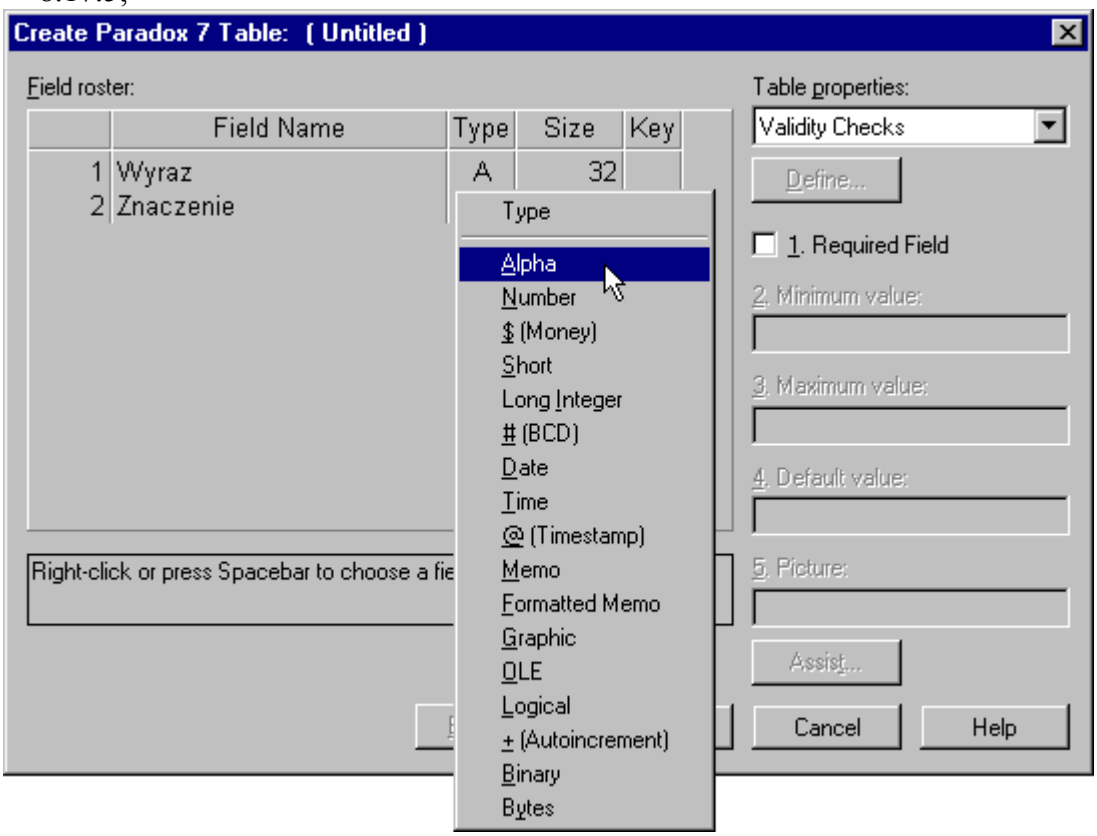
Rysunek 8.17.3. Widok wybranego polecenia Table (Tabela)

- Będąc w oknie **Create Table** (Tworzenie tabeli) wybierz rodzaj bazy, w której chcesz stworzyć tabelę (domyślnie ustawiony typ tabeli to PARADOX 7) – rysunek 8.17.4;



Rysunek 8.17.4. Widok okna Create Table (Tworzenie tabeli)

- Wybierz tabelę typu PARADOX 7 i zatwierdź wybór klawiszem ENTER;
- W wyświetlonym oknie **Create Paradox 7 Table** zdefiniuj potrzebne pola - rysunek 8.17.5;



Rysunek 8.17.5. Widok okna Create Paradox 7 Table z dwoma zdefiniowanymi polami

- Po zdefiniowaniu pól zapisz tabelę, wybierając klawisz z napisem **Save as...** (Zapisz jako).

Po tych czynnościach tabelę trzeba podłączyć do programu, za pomocą komponentów do obsługi baz danych. Komponenty te znajdują się, na zakładce **Data Access** i **Data Controls** palety komponentów.
Dla potrzeb tego programu skorzystamy z trzech komponentów, tj. **Query**, **DataSource** i **DBGrid**.

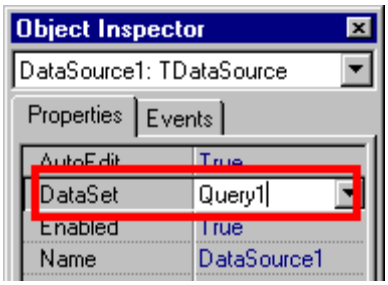
Query – jest komponentem reprezentującym zbiór danych, które są wynikiem zapytania SQL (ang. Standard Query Language – Standardowy Język Zapytań).

DataSource – jest komponentem realizującym połączenie pomiędzy zbiorami danych (tabelą) a innymi komponentami (np. **DBGrid**).

DBGrid – jest komponentem służącym do wyświetlania zawartości zbioru danych (np. tabeli).

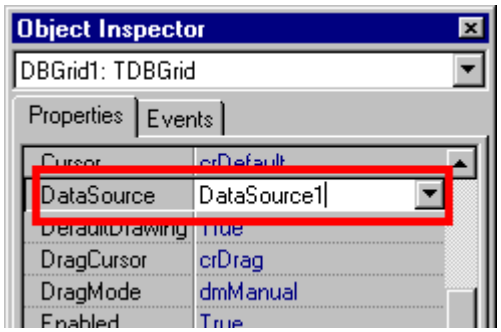
Połączenie komponentów DataSource, DBGrid oraz Query jest następujące:

- Połącz komponent **DataSource** (np. **DataSource1**) z komponentem **Query** (np. **Query1**) za pomocą właściwości **DataSet** komponentu **DataSource** – rysunek 8.17.6;



Rysunek 8.17.6. Widok właściwości DataSet komponentu DataSource1

- Połącz komponent **DBGrid** (np. **DBGrid1**) z komponentem **DataSource** (np. **DataSource1**) za pomocą właściwości **DataSource** komponentu **DBGrid** – rysunek 8.17.7.

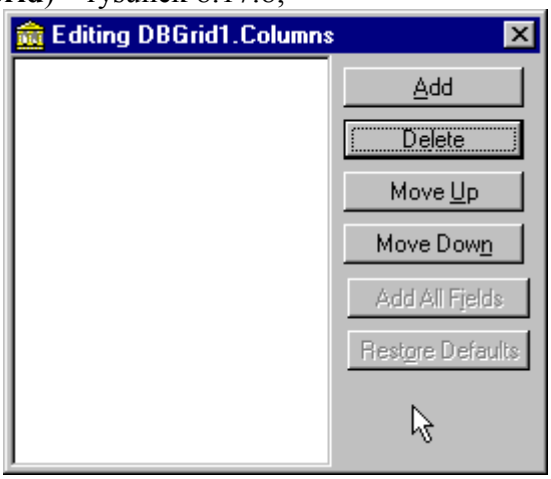


Rysunek 8.17.7. Widok właściwości DataSource komponentu DBGrid

- Komponenty zostały połączone.

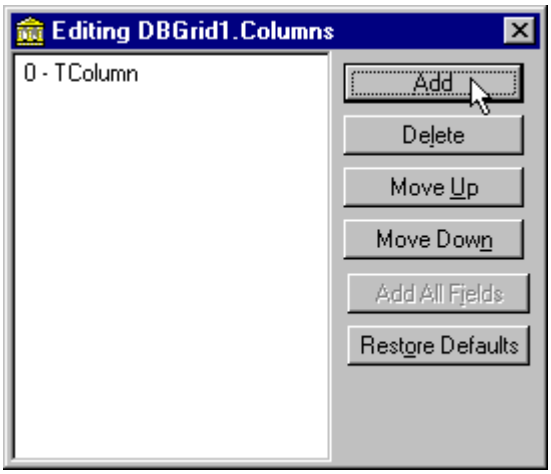
Nazwy kolumn w komponencie DBGrid:
Standardowo nazwy wyświetlane w kolumnach są pobierane z bazy. W celu zmiany opisu kolumn należy wykonać następujące czynności:

- Kliknij dwukrotnie na komponent **DBGrid** (umieszczony jest na formatce) , co spowoduje ukazanie się okna **Editing DBGrid1.Columns** (Edycja kolumn komponentu **DBGrid**) – rysunek 8.17.8;



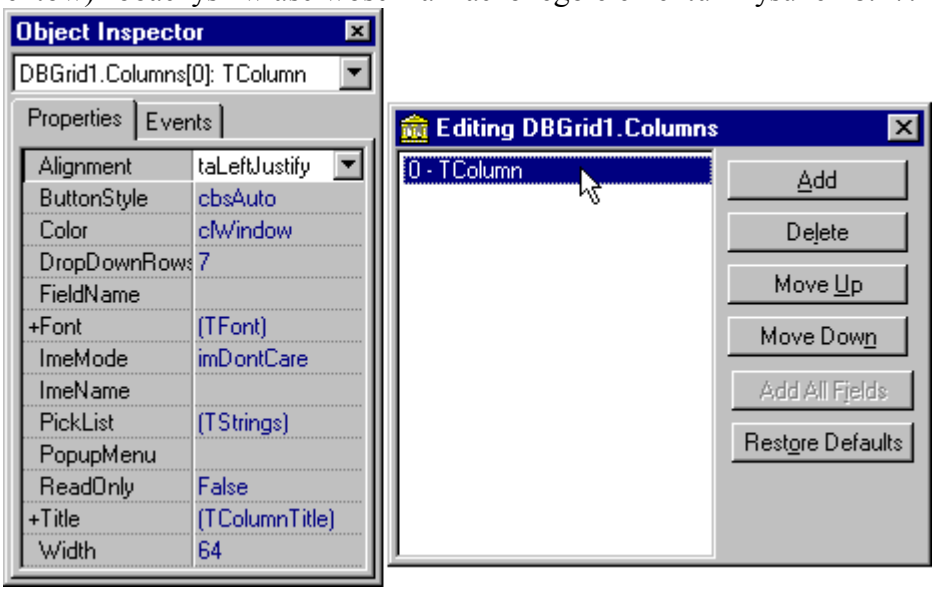
Rysunek 8.17.8. Widok okna Editing DBGrid1.Columns (Edycja kolumn komponentu DBGrid)

- Kliknij na klawisz z napisem **Add** (Dodaj), co spowoduje ukazanie się nowego elementu na liście – rysunek 8.17.9;



Rysunek 8.17.9. Widok nowego elementu na liście

- Zaznacz element na liście, w wyniku czego w oknie **Object Inspector** (Inspektora Obiektów) zobaczysz właściwości zaznaczonego elementu – rysunek 8.17.10;

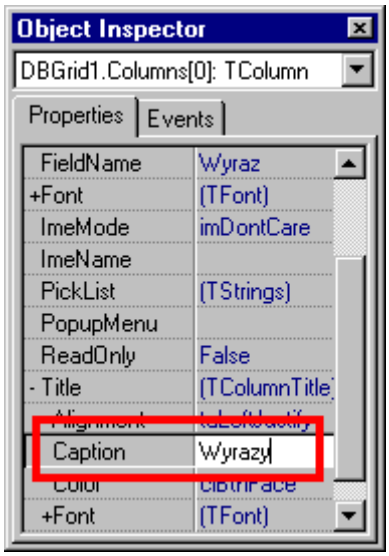


Rysunek 8.17.10. Widok właściwości wybranego elementu z listy

- Wpisz nazwę pola zgodną z nazwą kolumny w bazie we właściwości **FieldName** zaznaczonego elementu – rysunek 8.17.11;

Rysunek 8.17.11. Widok wpisanej nazwy pola dla wybranego elementu

- Rozwiń właściwość **Title** przez dwukrotne kliknięcie na tej nazwie;
- Wpisz we właściwości **Caption** tekst „Lista wyrazów” – rysunek 8.17.12;



Rysunek 8.17.12. Widok wpisanej tekstu „Wyrazy”

- Z następnymi elementami listy postępuj tak samo.

Połączenie komponentów DataSource, DBEdit oraz Query jest następujące:

- Połącz komponent **DataSource** (np. **DataSource1**) z komponentem **Query** (np. **Query1**) za pomocą właściwości **DataSet** komponentu **DataSource** (np. **DataSource1**) przypisując tej właściwości komponent **Query** (np. **Query1**);
- Połącz komponent **DBEdit** (np. **DBEdit1**) z komponentem **DataSource** (np. **DataSource1**) za pomocą właściwości **DataSource** komponentu **DBEdit** (np. **DBEdit1**) przypisując tej właściwości komponent **DataSource** (np. **DataSource1**);
- We właściwości **DataField** komponentu **DBEdit** (np. **DBEdit1**) wpisz nazwę pola (kolumny);
- Przy kolejnych komponentach **DBEdit** postępuj tak samo.

Umieszczenie kodu programu w drugim module (plik Unit2):

- Kliknij dwukrotnie na formatce i w wygenerowanym zdarzeniu **OnCreate** wpisz kod:

```
procedure TForm2.FormCreate(Sender: TObject);
begin
  Edit1.Text:= ''; // Wyczyszczenie zawartości komponentu EDIT.
  Edit2.Text:= '';
end;
```

- Zadeklaruj zmienną logiczną „okForm” w sekcji **public** (Publicznej):

```
private
  { Private declarations }
public
  { Public declarations }
  okForm :Boolean; // Zadeklarowanie zmiennej logicznej
end;
```

- W zdarzeniu **OnShow** umieść linię „okForm:= FALSE”:

```
procedure TForm2.FormShow(Sender: TObject);
begin
  // FormShow
  okForm:= FALSE; // Przypisanie zmiennej "okForm" wartości FALSE (Fałsz)
end;
```

- Kliknij dwukrotnie na klawisz z napisem „OK” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
  {
   Przypisanie zmiennej "okForm" wartości TRUE (Prawda),
   gdy zostanie naciśnięty klawisz OK, tzn. nowe słówko zostanie dodane do bazy.
  }
  okForm:= TRUE;
  Close; // Zamknięcie formatki
end;
```

Umieszczenie kodu programu w module głównym (plik Unit1):

- Wpisz w wygenerowanym zdarzeniu **OnShow** kod odpowiedzialny za wyświetlenie zawartości bazy (tabeli):

```
procedure TfrmForm1.FormShow(Sender: TObject);
begin
 // Wyświetlenie zawartości bazy słownika

  Query1.Close; // Zamknięcie komponentu, w celu umożliwienia wymiany instrukcji SQL
  Query1.SQL.Clear; // Czyszczenie zapytania SQL

  // Zapytanie SQL
  Query1.SQL.Add('SELECT * FROM „slownik.db” ORDER BY Wyraz');
  {
   SELECT * FROM „slownik.db” ORDER BY Wyraz
   Wyświetla zawartość tablicy SLOWNIK.DB posortowaną
   alfabetycznie według kolumny "Wyraz"

   SELECT - Używany jest do formułowania zapytań do
          bazy danych w celu uzyskania informacji.

   * - Oznacza wyświetlenie wszystkich kolumn z danej bazy.
      Gdyby była napisana nazwa kolumny (np. wyraz) zamiast
      Gwiazdki, to wyświetlona zostałaby kolumna o nazwie "Wyraz".

   FROM - określa, z jakiej tablicy lub, z jakich tablic są pobierane dane.
         W naszym przypadku jest to tablica o nazwie "slownik.db".

   ORDER BY - klauzula ta służy do sortowania według wybranej kolumny.
            W naszym przykładzie jest to kolumna "Wyraz".
  }
```

```
  Query1.Open; // Otwarcie bazy

  // Wyświetla ilość wierszy (rekordów) w tabeli.
  Label5.Caption:= 'Ilość słów: '+IntToStr(Query1.RecordCount);

  {
  Tworzenie bazy robimy przez wstawienie kodu pod jakiś klawisz:
  Query1.Close;
  Query1.SQL.Clear;
  Query1.SQL.Add('CREATE TABLE „slownik.db” ('+
                 'Wyraz CHAR(32), '+
                 'Znaczenie CHAR(255), '+
                 'ID CHAR(20))');
  Query1.ExecSQL;
  }
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Dodaj” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Dodanie nowego słówka do bazy.

  // Wyświetlenie nagłówka okna edycyjnego.
  Form2.Caption:= 'Dodaj';

  // Wyczyszczenie zawartości komponentu EDIT w oknie edycyjnym.
  Form2.Edit1.Text:= '';

  // Wyczyszczenie zawartości komponentu EDIT w oknie edycyjnym.
  Form2.Edit2.Text:= '';

  // Wywołanie nowej formatki - okna edycyjnego do wprowadzenia nowego słówka.
  Form2.ShowModal;

  {
   Sprawdzenie czy został naciśnięty klawisz OK,
   jeżeli tak, dodaj słówko do tabeli.
  }
  if (Form2.okForm = TRUE) then
  begin
    // Dodanie nowego słówka (nowego rekordu).
    Query1.Close; // Zamknięcie komponentu, w celu umożliwienia wymiany instrukcji SQL
    Query1.SQL.Clear;

    // INSERT INTO - Wstawia jeden lub więcej wierszy do tabeli.
    Query1.SQL.Add('INSERT INTO "slownik.db" ('+
                   'Wyraz, '+
```

```
                'Znaczenie, '+
                'ID) '+
            'VALUES (:p0, :p1, :p2)');

    // Pole WYRAZ
    Query1.Params[0].AsString:= Trim(Form2.Edit1.Text);
    {
     Właściwość Params jest tablicą numerowaną od zera, którą
     można wykorzystać do przypisania wartości w czasie
     wykonywania programu. Zamiast właściwości Params można
     zastosować właściwość ParamByName(), np.
      Query1.ParamByName('Wyraz').AsString:= 'think';
    }

    // Pole "Znaczenie".
    Query1.Params[1].AsString:= Trim(Form2.Edit2.Text);

    // Pole identyfikacyjne
    Query1.Params[2].AsString:= DateTimeToStr(Now);
    {
     DateTimeToStr(Now) - Zwraca datę i czas.
     DateTimeToStr() - Dokonuje konwersji daty i czasu na tekst (string).
     Now - Zwraca aktualną datę i czas.

     Pole identyfikacyjne, za pomocą, którego będzie
     można poprawiać lub usuwać dane słówko.
     W tym celu wykorzystana została data i godzina.
     Parametry te są różne dla każdego słówka.
    }

    Query1.ExecSQL; // Wykonanie zapytania SQL.

    FormShow(Sender);
    {
     FormShow(Sender); - Wywołanie funkcji odczytujące tabelę w celu
                         odświeżenia wyświetlanej informacji.
    }
  end;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Edycja” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  txtID: String; // Zadeklarowanie zmiennej "txtID".
begin
  // Edycja wybranego wiersza (rekordu).

  // Przekazanie wartości z pola identyfikacyjnego "ID" do zmiennej "txtID".
```

```
  txtID:= '';
  txtID:= Query1.FieldByName('ID').AsString;

  // Wyświetlenie nagłówka okna edycyjnego.
  Form2.Caption:= 'Edycja';

  // Odczytanie wiersza z komponentu DBGrid (wyraz). Kolumna 0.
  Form2.Edit1.Text:= DBGrid1.Fields[0].AsString;

  // Odczytanie wiersza z komponentu DBGrid (znaczenie). Kolumna 1.
  Form2.Edit2.Text:= DBGrid1.Fields[1].AsString;

  // Wywołanie nowej formatki - okna edycyjnego do wprowadzenia nowego słówka.
  Form2.ShowModal;

  {
   Sprawdzenie czy został naciśnięty klawisz OK,
   jeżeli tak dodaj słówko do tabeli.
  }
  if (Form2.okForm = TRUE) then
  begin
    Query1.Close; // Zamknięcie komponentu, w celu umożliwienia wymiany instrukcji SQL
    Query1.SQL.Clear;
    Query1.SQL.Add('UPDATE "slownik.db" SET Wyraz = :p0, Znaczenie = :p1 '+
              'WHERE ID = :p2');
    Query1.Params[0].AsString:= Trim(Form2.Edit1.Text); // Wyraz
    Query1.Params[1].AsString:= Trim(Form2.Edit2.Text); // Znaczenie
    Query1.Params[2].AsString:= Trim(txtID); // Pole identyfikacyjne
    Query1.ExecSQL; // Wykonanie zapytania SQL.

    FormShow(Sender);
    {
     FormShow(Sender); - Wywołanie funkcji odczytujące tabelę w celu
                    odświeżenia wyświetlanej informacji.
    }
  end;
end;
```

- Kliknij dwukrotnie na klawisz z napisem „Usuń” i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button3Click(Sender: TObject);
var
  txtID: String; // Zadeklarowanie zmiennej "txtID".
begin
  // Usuwa zaznaczony wiersz (rekord).

  // Przekazanie wartości z pola identyfikacyjnego "ID" do zmiennej "txtID".
  txtID:= '';
  txtID:= Query1.FieldByName('ID').AsString;
```

```
  Query1.Close; // Zamknięcie komponentu, w celu umożliwienia wymiany instrukcji SQL
  Query1.SQL.Clear;
  Query1.SQL.Add('DELETE FROM "slownik.db" WHERE ID = :p0');

  // Wykorzystanie identyfikatora do usunięcia wybranego rekordu.
  Query1.Params[0].AsString:= txtID;

  Query1.ExecSQL; // Wykonanie zapytania SQL.
  FormShow(Sender);
  {
   FormShow(Sender); - Wywołanie funkcji odczytujące tabelę w celu
                       odświeżenia wyświetlanej informacji.
  }
end;
```

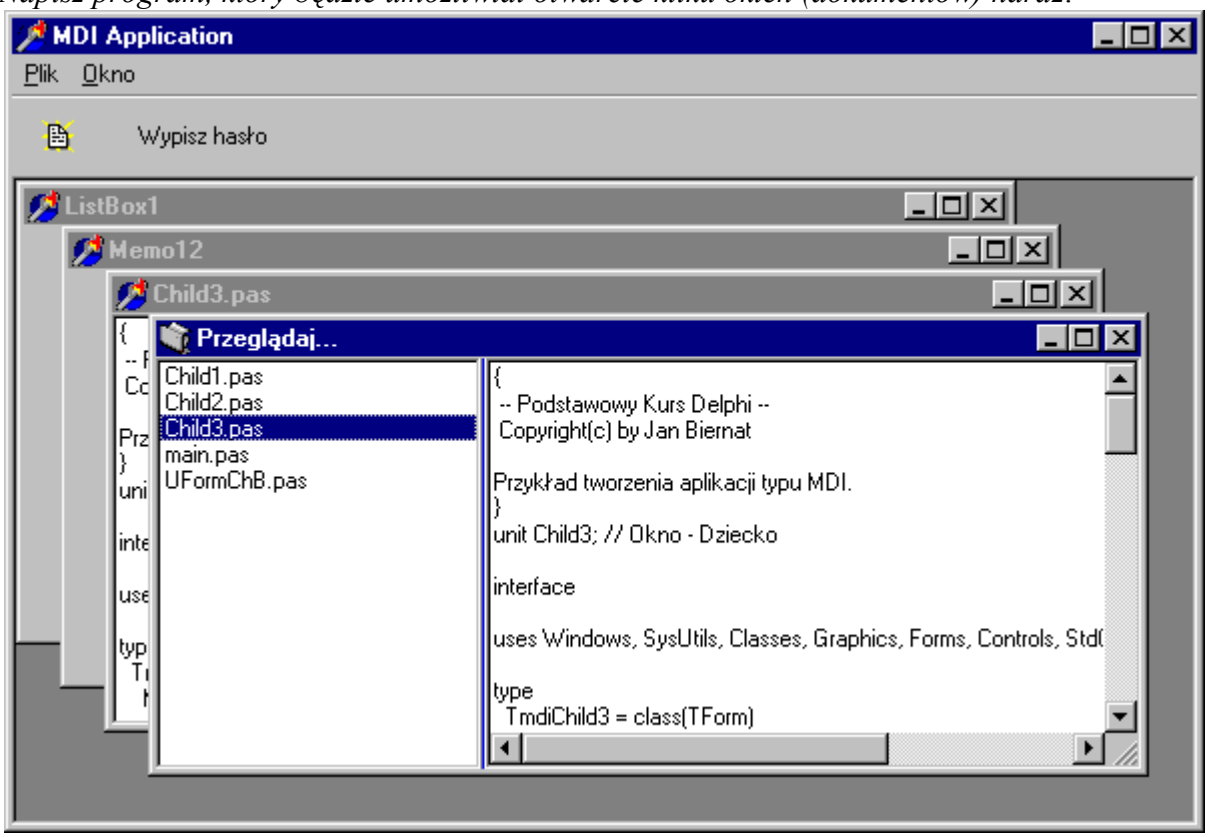
- Kliknij dwukrotnie na komponent **Edit** (znajdujący się na formatce) i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  // Wyszukanie wyrazu (słówka).
  Query1.Close; // Zamknięcie komponentu, w celu umożliwienia wymiany instrukcji SQL
  Query1.SQL.Clear;
  Query1.SQL.Add('SELECT * FROM "slownik.db" '+
              'WHERE UPPER(Wyraz) LIKE :p_Wyraz '+
              'ORDER BY Wyraz');
  {
   WHERE - Po słowie WHERE występuje predykat, który składa
         się z jednego lub więcej wyrażeń.
         W tym przypadku UPPER(Wyraz) LIKE :p_Wyraz.

   UPPER() - Konwertuje ciąg znaków na ciąg znaków pisanych dużymi literami.

   LIKE - Wyrażenie to pozwala nam na poszukiwanie określonego ciągu znaków.
  }


  {
   Przekazanie ciągu znaków jako parametru
   według którego nastąpi wyszukiwanie (pole Wyraz).
   % - Pozwala na wyświetlenie wszystkich wyrazów
       zaczynających się od litery np. A, a dzięki
       operatorowi "%" dalsze litery nie są brane pod uwagę.
  }
  Query1.Params[0].AsString:= UpperCase(Trim(Edit1.Text))+'%';
  Query1.Open; // Otwarcie bazy
end;
```

- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Ćwiczenie 8.18. MDI (aplikacja wielodokumentowa)

Napisz program, który będzie umożliwiał otwarcie kilku okien (dokumentów) naraz.



Rysunek 8.18.1. Widok programu

Przedstawiony przykład znajduje się w katalogu D7CW\MDI.

Opis:
Aplikacje o interfejsie wielodokumentowym mogą jednocześnie otwierać i wyświetlać kilka dokumentów. Aplikacja typu MDI (ang. Multiple Document Interface) dzieli się na okno główne (zwane rodzicem) i okna podrzędne (zwane dziećmi). Każde okno dziecka jest oddzielnym dokumentem działającym niezależnie do innych okien dzieci. Okno dziecka posiada własne menu, które jest niezależne do innych menu. Menu to tworzy się tak samo jak dla formatki, oddzielnie dla każdego okna dziecka. Wszystkie okna dzieci są wyświetlane na wspólnym obszarze rodzica i nie mogą wyjść poza ten obszar. Wyjście poza ten obszar spowoduje chowanie się okna dziecka. Pomimo wyświetlania dużej ilości okien dzieci, użytkownik może pracować w danej chwili tylko na jednym oknie dziecka.

Sposób tworzenia aplikacji MDI:

- Wybierz menu **File/New** (Plik/Nowy);
- Wybierz zakładkę **Projects** (Projekty);
- Wybierz projekt o nazwie **MDI application**;
- Zatwierdź klawiszem **OK**;
- W oknie **Select Directory** (Wybierz katalog) wybierz katalog, do którego zostanie zapisany wygenerowany kod;
- Po wybraniu katalogu zatwierdź wybór klawiszem **OK**;
- Naciśnij klawisz funkcyjny **F9**, w celu uruchomienia programu.

Po tych czynnościach aplikacja typu MDI jest już wygenerowana.

Poniżej są zamieszczone fragmenty kodu, które umożliwiają dodanie nowego okna dziecka: Deklaracja okna dziecka znajduje się w sekcji **Private** (Prywatnej).

```
  private
    { Private declarations }
    procedure CreateMDIChild1(const Name: string);
```

Natomiast moduł deklarujemy po słowie **uses**. W tym przypadku jest to okno dziecka.

```
  uses Child1;
```

Następnie umieszczona zostaje funkcja wywołująca okno dziecka:

```
procedure TMainForm.CreateMDIChild1(const Name: string);
var
  Child: TmdiChild1;
begin
  { create a new MDI child window }
  Child := TmdiChild1.Create(Application);
  Child.Caption := Name;
end;
```

Zamknięcie okna dziecka jest możliwe dzięki wywołaniu funkcji:

```
procedure TMainForm.FileCloseItemClick(Sender: TObject);
begin
  // Zamknięcie aktywnego okna dziecka
  if ActiveMDIChild <> nil then
    ActiveMDIChild.Close;
end;
```

Uruchomienie funkcji znajdującej się w oknie dziecka za pomocą klawisza znajdującego się w oknie głównym (oknie rodzica) wygląda następująco:

```
procedure TMainForm.sbWypiszHasloClick(Sender: TObject);
begin
  // Uruchamia funkcję z menu, które znajduje się w oknach dzieci

  // Uruchomienie Funkcji "ListBoxNapis1Click(Sender);" w Dziecku 1 (Child1)
  if ActiveMDIChild is TmdiChild1 then
    TmdiChild1(ActiveMDIChild).ListBoxNapis1Click(Sender);
end;
```

Uruchomienie funkcji w innym oknie dziecka (okno posiada inną nazwę np. TmdiChild2) wygląda tak samo, z jedyną różnicą. Różnica ta polega na zmianie nazwy okna TmdiChild1 na TmdiChild2. Nazwę okna formatki wpisujemy we właściwości **Name** wybranej formatki.

Uruchomienie funkcji znajdującej się w oknie rodzica z okna dziecka wygląda następująco:

```
procedure TmdiChild1.bOtworzPlikClick(Sender: TObject);
begin
  // Uruchamia opcję znajdującą się w oknie rodzica z poziomu okna dziecka.
  MainForm.FileOpenItemClick(Sender);
```

```
  {
  MainForm.FileOpenItemClick(Sender); - Uruchomienie funkcji
  MainForm - Nazwa okna rodzica
  FileOpenItemClick(Sender) - Nazwa funkcji znajdującej się w oknie rodzica
  }
end;
```

Jak na aplikację wielodokumentową przystało możliwe jest otwarcie kilku okien takich samych lub różnych (np. wybierz dwa razy opcję „Nowy (ListBox)" – spowoduje pojawienie się dwóch takich samych okien). Istnieje jednak potrzeba, aby jakieś okno (np. do przeglądania plików) było otwarte tylko raz, niezależnie od ilości wybierania danej opcji (w tym przykładzie jest to opcja „Przeglądaj..."). Wybranie tej opcji ponownie (po wywołaniu okna „Przeglądaj...") spowoduje uaktywnienie tegoż okna z automatycznym przesunięciem go na plan główny. Funkcja, która to wykonuje wygląda następująco:

```
procedure TMainForm.FilePrzegldajClick(Sender: TObject);
const
// Zadeklarowanie stałej określającej nazwę okienka służącego do przeglądania plików.
  txtPrzeglad ='Przeglądaj...';
var
          TT: Integer; // Zadeklarowanie zmiennej liczbowej
  okCzyIstnieje: Boolean; // Zadeklarowanie zmiennej logicznej
begin
  // Uruchomienie przeglądania plików.
  okCzyIstnieje:= FALSE; // Przypisanie zmiennej wartości FALSE (fałsz)

  // Sprawdzenie, które okno służy do przeglądania plików.
  for TT:= MDIChildCount - 1 downto 0 do
    if (MDIChildren[TT].Caption = txtPrzeglad) then
    begin
      // Gdy okno zostanie znalezione, to uaktywnij go i daj na pierwszy plan.
      MDIChildren[TT].BringToFront; // Przesuwa okna na pierwszy plan
      MDIChildren[TT].WindowState:= wsNormal; // Powoduje ukazanie się okna

       // Przypisanie zmiennej 'okCzyIstnieje' wartości TRUE jeżeli okno do przeglądania
       // plików zostało znalezione (jest uaktywnione).
       okCzyIstnieje:= TRUE;
    end;


  // Uruchomienie okna do przeglądania plików, jeżeli zmienna 'okCzyIstnieje'
  // równa się FALSE.
  if (okCzyIstnieje = FALSE) then CreateMDIChildPrzegladaj(txtPrzeglad);
end;
```

Tworząc menu w oknie rodzica musimy uważać, żeby menu stworzone w oknie dziecka nie zastąpiło menu w oknie rodzica (chyba, że to jest konieczne). Zapobiec temu można przez nadanie poszczególnym elementom głównym menu w oknie dziecka numeru różnego od zera.

Numer ten należy wpisać we właściwości **GroupIndex**. Dzięki temu po wywołaniu okna dziecka nastąpi połączenie menu (okna rodzica) z menu (okna dziecka).

9. Ćwiczenia do samodzielnego wykonania

Ćwiczenie 1:
Napisz program, który będzie wyświetlał tablicę kodów ASCII.

Ćwiczenie 2:
Napisz program, który będzie odtwarzał pliki dźwiękowe pobrane z listy.

Ćwiczenie 3:
Jaka jest różnica pomiędzy komponentem Panel, a Shape – zaprezentuj.

Ćwiczenie 4:
Napisz program, który będzie automatycznie zmieniał imiona i nazwiska tak, aby zaczynały się z dużej litery, natomiast tytuły naukowe pisał z małej litery.

Ćwiczenie 5:
Napisz program do obliczania równania kwadratowego i X1, X2.

Ćwiczenie 6:
Napisz program, który poda nazwę dnia na podstawie daty.

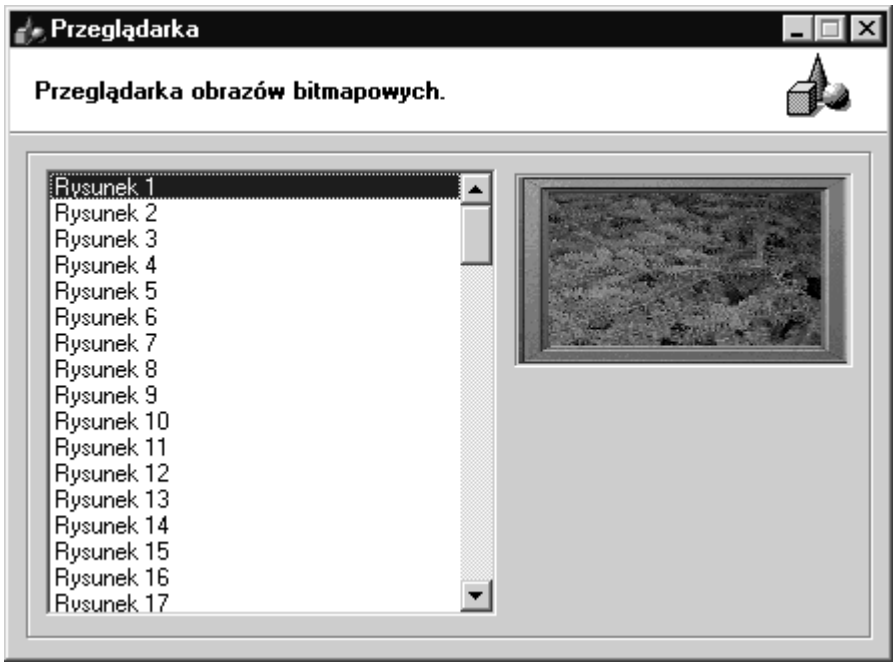
Ćwiczenie 7:
Napisz program, który będzie losował liczby i konwertował je na słowa.

Ćwiczenie 8:
Napisz program do nauki tabliczki mnożenia.

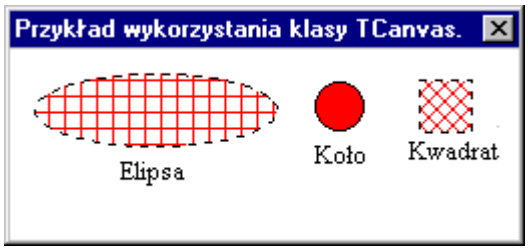
Ćwiczenie 9:
Umieść obrazek w komponencie StringGrid (zdarzenie StringGrid1DrawCell) w wierszu 5 i kolumnie 6.
Skorzystaj z poniższego przykładu:

```
if (Col = 1) and (Row = 3) then
begin
  StringGrid1.Canvas.Draw(Rect.Left+2, Rect.Top+2, Image1.Picture.Graphic);
end;
```

Ćwiczenie 10:
Napisać przeglądarkę obrazków bitmapowych. Rysunek przedstawia wygląd przykładowej przeglądarki.



Ćwiczenie 11:
Napisać program, który losowo będzie rysował figury na formie i podpisywał narysowane figury. Rysunek przedstawia przykładowy program:



Ćwiczenie 12:
Napisz program do przechowywania informacji dotyczących: - nazwiska; - imienia; - adresu; - telefonu. Do pisania tego programu wykorzystaj komponent StringGrid. Program ten ma umożliwiać dodawanie, edycję, usuwanie, odczytywanie i zapisywanie oraz wyszukiwanie i sortowanie danych.

Ćwiczenie 13:
Napisz program, który będzie podawał ilość dni pomiędzy dwoma dowolnymi datami.

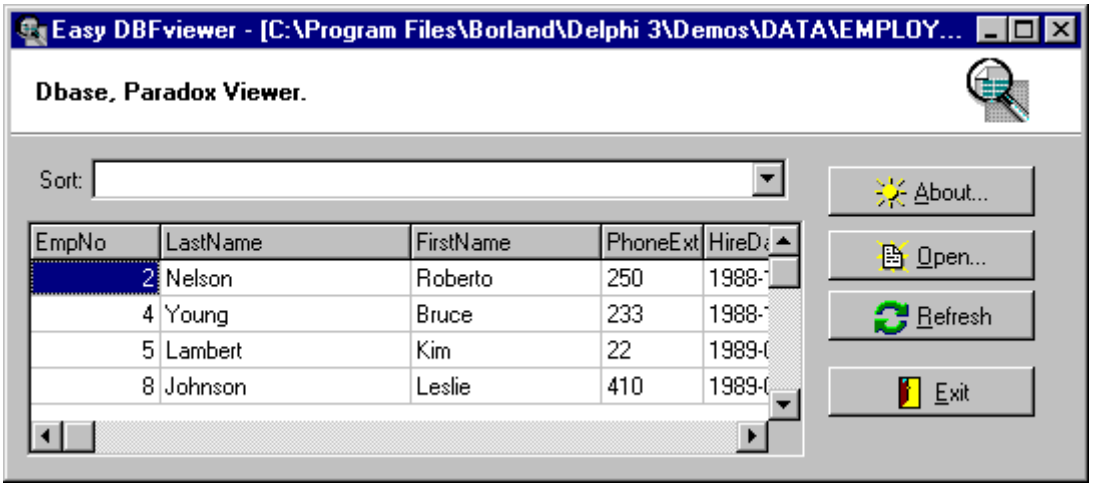
Ćwiczenie 14:
Napisz program, który poda na podstawie daty urodzenia danej osoby następujące informacje: - wiek osoby (brać pod uwagę rok przestępny); - dni imienin; - dzień urodzin oraz nazwę dnia, w którym się urodził.

Ćwiczenie 15:
Napisz program, który będzie liczył średnią z podanych ocen oraz będzie podawał ilość ocen pozytywnych i negatywnych.

Ćwiczenie 16:
Napisz program, który będzie automatycznie numerował nazwy plików przy zapisywaniu.

Ćwiczenie 17:
Napisz program „Budzik” lub „Przypominaj o świętach”.

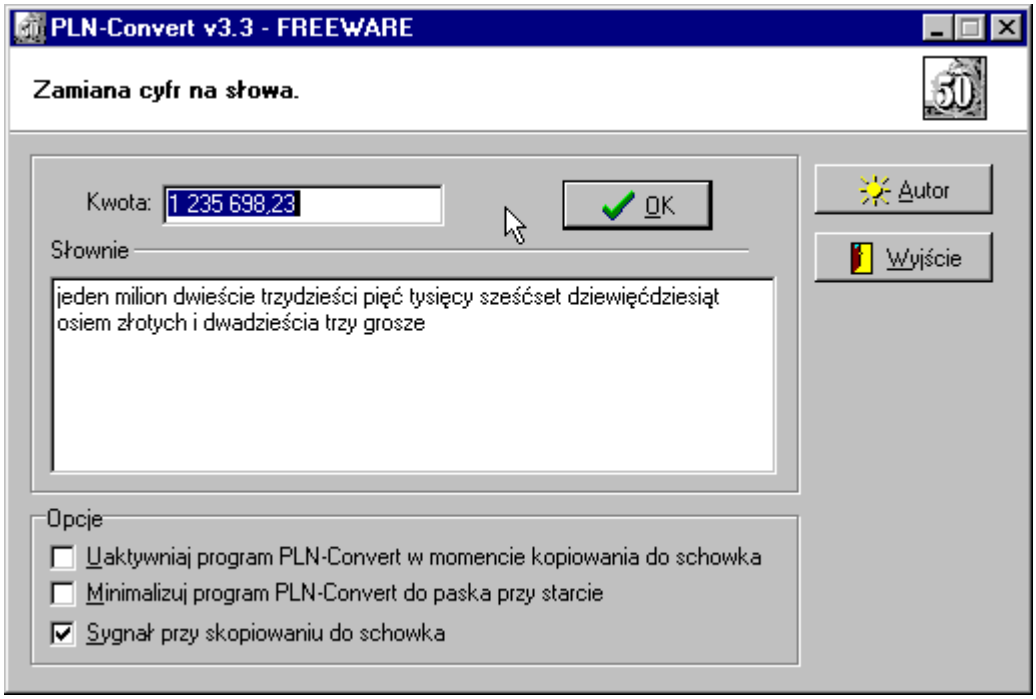
Ćwiczenie 18:
Napisz przeglądarkę do plików baz danych DBase i Paradox’a z możliwością wczytania pliku przez podanie go jako parametr przy wywoływaniu tego programu. Rysunek przedstawia przykładowy program:



Ćwiczenie 19:
Napisz program do odczytywania ikon, wykorzystując dynamiczne tworzenie komponentów. Ikony mają być w kilku kolumnach. Rysunek poniższy przedstawia przykładowy program.

Ćwiczenie 20:

Napisz program, który będzie zamieniał liczby na słowa przez schowek.



Ćwiczenie 21:
Stwórz bibliotekę zawierającą dwie funkcje (tj. WyswietlDate, GodzinaJest) oraz zaprezentuj jej działanie.

Ćwiczenie 22:
Napisz program „Przypominaj o ważnych datach”, który będzie rozróżniał rok przestępny, wykrywał ostatni dzień w miesiącu oraz poszczególne dni.