

Spis treści

Wstęp	3
1. Środowisko programistyczne	4
1.1 Komponenty	4
1.2 Formatki	5
1.3 Inspektor obiektów	6
1.4 Edytor kodu	8
2. Typy plików w Delphi	11
3. Elementy Pascala	12
3.1 Komentarze	12
3.2 Wybrane typy	12
3.3 Zmienne	13
3.4 Stałe	13
3.5 Instrukcje warunkowe IF...Then / IF...Then...Else	14
3.6 Instrukcja wiążąca With...Do	15
3.7 Pętle For...To/Downto... Do, While...Do, Repeat...Until	15
3.8 Mechanizmy obsługi wyjątku	17
3.9 Opis wybranych zdarzeń	18
3.10 Nazwa aplikacji (programu)	20
3.11. Lista wybranych zmiennych globalnych	21
3.12. Operatory logiczne	22
4. Pierwszy program	22
5. Ćwiczenia z podstawowych komponentów	23
5.1 ComboBox	23
5.2 Edit, BitBtn	26
5.3 Label	28
5.4 komponent ListBox	30
5.4.1 ListBox	30
5.4.2 ListBox	34
5.4.3 ListBox	38
5.4.4 ListBox	40
5.5 CheckListBox	44
5.6 Memo	45
5.7 Image	47
5.8 ScrollBox	48
5.9 ProgressBar	49
5.10 StatusBar	49
5.11 Timer	50
5.12 RadioButton, GroupBox i CheckBox	51
5.13 TrackBar	53
5.14 MediaPlayer	54
5.15 Menu	55
5.16 PopupMenu	57
5.17 StringGrid	58
5.18 TreeView	62
5.19 Wewnętrzna lista	65
5.20 Okna informacyjne	67

5.21 Okna służące do zadawania pytań	69
5.22 Okna dialogowe	71
5.23 Obsługa kilku komponentów	74
5.24 UpDown	76
5.25 ScrollBar	77
5.26 Splitter	78
5.27 Dynamiczne tworzenie komponentów	79
5.28 Wczytanie czcionek do listy ComboBox i ListBox	84
5.29 Instalacja nowych komponentów	86
5.30 Dymki (Podpowiedzi)	87
5.31 MDI (aplikacja wielodokumentowa)	88
5.32 Wczytanie pliku przez podanie jego nazwy jako parametr	91
5.33 Uruchomienie innego programu z poziomu aplikacji	92
5.34 Pozycja kursora myszy	93
5.35 Zamiana znaków w tekście	94
5.36 Grafika	96
5.37 funkcja słownie	99
5.38 Wprowadzenie do baz danych	105
5.39 Wyświetlenie tabeli w komponencie StringGrid	113
5.40 Sortowanie w komponencie StringGrid	118
5.41 Alarm	122
5.42 Standardowe animacje w systemie Windows	125
5.43 Dodawanie dni do daty	126
5.44 Nazwy dni i miesiące	132
5.45 PageControl (Zakładki)	135
5.46 Rysowanie na ograniczonym obszarze	137
5.47 Memo do wyświetlania krótkiej pomocy	139
5.48 Wątki	142
5.49 Wydruki	147
5.50 Wyświetlenie nazw kolumn z bazy	159
5.51 Wyselekcjonowanie liczb z tekstu	160
5.52 Gauge	163
5.53 Chart – wykresy	165
5.54. Liczenie plików	173
5.55. Okno InputBox	176
5.56. Plik rekordowy	177
5.57. Odliczanie czasu w tył	181
5.58. Zmiana wielkości liter	184
5.59. Szukanie danych w komponencie StringGrid	187
5.60. Polskie znaki (Ogonki)	191
6. Ćwiczenia do samodzielnego wykonania	194

Wstęp

Książka obejmuje ćwiczenia z programu Borland Delphi i jest przeznaczona dla szerokiego grona użytkowników znających obsługę systemu Windows, którzy chcieliby w szybki i przyjemny sposób rozpocząć naukę programowania.

Borland Delphi jest 32 bitowym środowiskiem programistycznym dającym możliwość obiektowego i wizualnego projektowania aplikacji, przez co zaliczane jest do narzędzi typu RAD (Rapid Application Development) – co oznacza szybkie tworzenie aplikacji.

Te walory przyczyniają się do tworzenia programów w sposób przyjemny i szybki. Programy zrobione za pomocą Delphi 3.0 są programami 32 bitowymi, przez co pracują szybciej od swoich 16 bitowych odpowiedników stworzonych za pomocą Delphi 1.0 lub Pascala.

Wszystkie przedstawione ćwiczenia dotyczą Delphi w wersji 3.0 lub wyższej i znajdują się na dołączonej do książki dyskietce. Znajduje się tam również kilka nowych komponentów (między innymi obsługa rysunków jpeg, wybór katalogu), które można zainstalować.

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

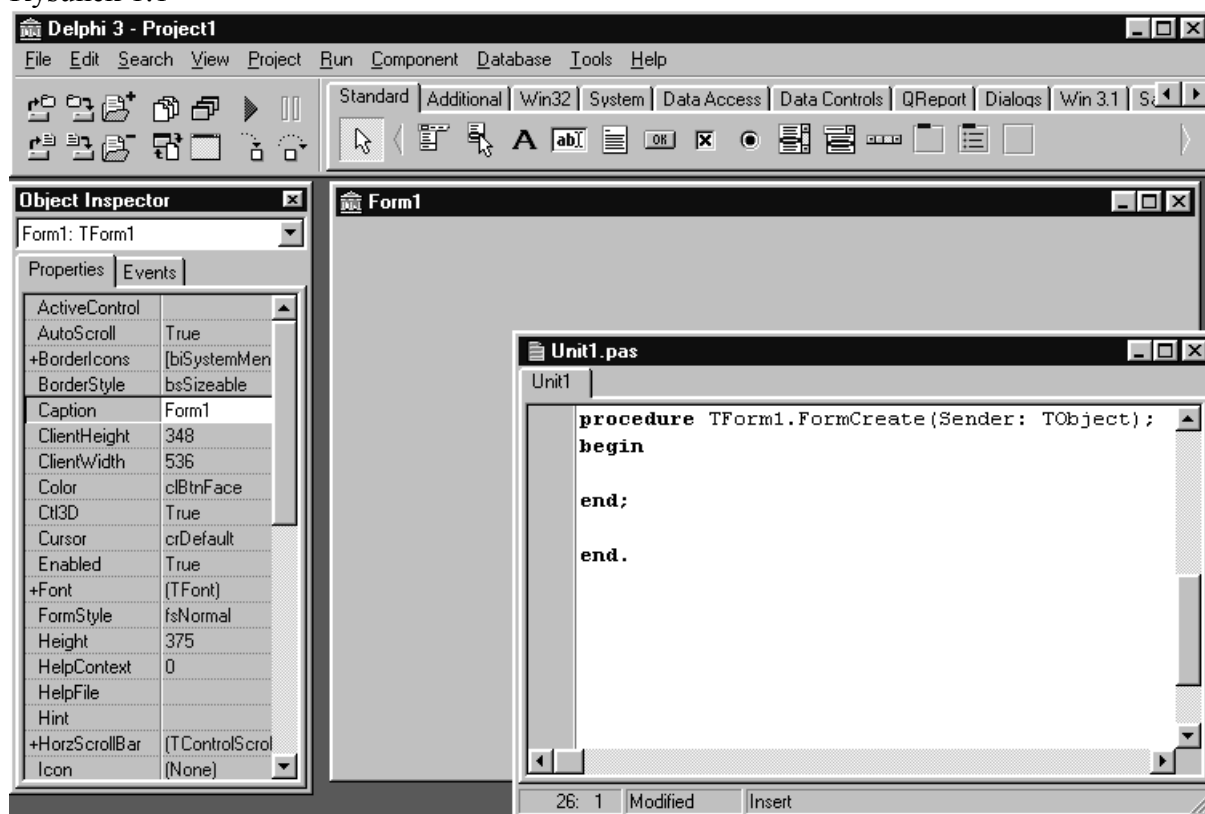
Autor

Książkę tę dedykuję swojej mamie

1. Środowisko programistyczne

Po uruchomieniu Delphi zobaczymy zintegrowane środowisko przedstawione na rysunku 1.1..

Rysunek 1.1



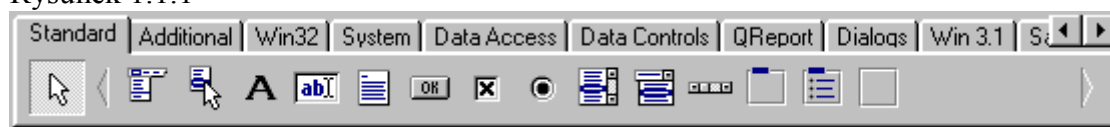
Środowisko to składa się z następujących elementów:

- Komponentów;
- Formatki (Form1);
- Inspektora obiektów;
- Edytora kodu;
- Pasków narzędzi;
- I wielu innych.

1.1 Komponenty

Komponenty w Delphi są bardzo dużym ułatwieniem w czasie tworzenia aplikacji. Dzięki nim można znacząco ograniczyć pisanie kodu, przez co programy posiadają mniej błędów. Komponenty umieszczone są z prawej strony poniżej paska menu. Rysunek 1.1.1 przedstawia paletę komponentów.

Rysunek 1.1.1



W celu umieszczenia komponentu na formatce należy wykonać następujące czynności:

- Wybrać komponent z palety komponentów (np. **Edit** z karty **Standard**). Po wskazaniu komponentu wskaźnikiem myszy zostanie on wyróżniony – rysunek 1.1.2.

Rysunek 1.1.2



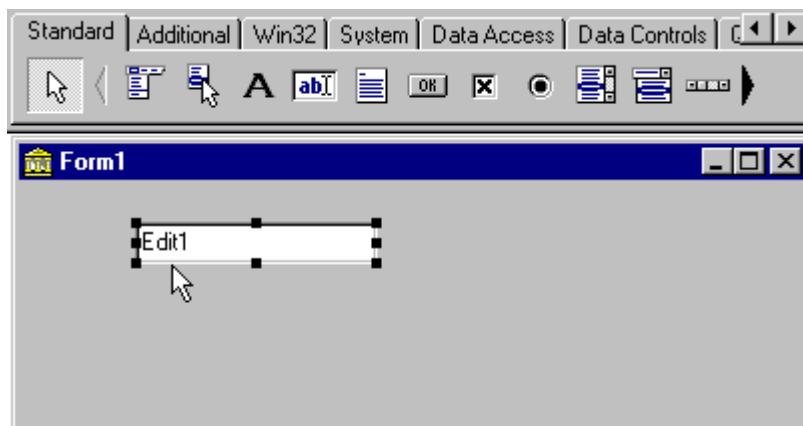
- Wyboru komponentu dokonujemy przez naciśnięcie(kliknięcie) na dany komponent, co jest przedstawione na rysunku 1.1.3.

Rysunek 1.1.3



- Po tych czynnościach wystarczy kliknąć na formatce w miejscu w którym ma się znaleźć wybrany komponent lub dwukrotnie kliknąć (szybko) na wybranym komponencie, co spowoduje umieszczenie tego komponentu na formatce. Taką sytuację ilustruje rysunek 1.1.4.

Rysunek 1.1.4



Zaznaczany automatycznie jest tylko jeden komponent, który został umieszczony na formatce – rysunek 1.1.4. Jeżeli na formatce jest więcej komponentów, to zaznaczenie przechodzi na następny komponent umieszczany na formatce.

Komponenty znajdujące się na formatce można zaznaczać przez kliknięcie na dany komponent, co spowoduje zaznaczenie wybranego komponentu 8 uchwyty (kwadracikami) – rysunek 1.1.4. Komponent oznaczony możemy przesuwać (przytrzymując wciśnięty lewy klawisz myszy), rozciągać i zmieniać jego właściwości.

1.2. Formatki

Formatki są podstawowym elementem Delphi na którym umieszczane są komponenty. W momencie uruchomienia Delphi formatka (ang. form) jest tworzona automatycznie stając się

oknem głównym naszej aplikacji o nazwie **Form1**. Wygląd formatki przedstawia rysunek 1.2.1.

Rysunek 1.2.1

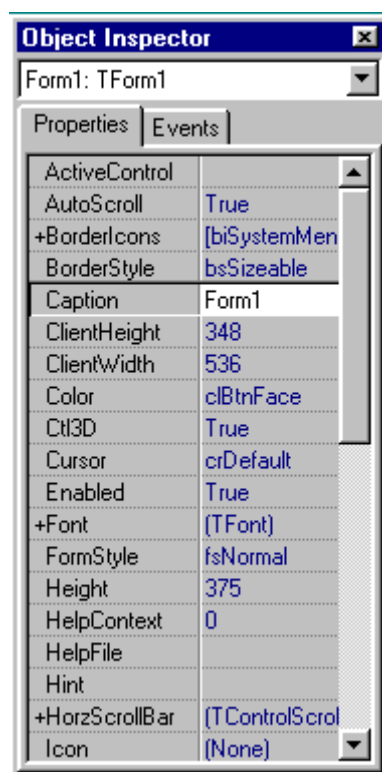


Wywołanie formatki robimy pisząc **Nazwa_formatki.ShowModal;**, np. nazwa formatki jest **Form2** to wywołujemy ją pisząc „**Form2.ShowModal;**”.

1.3. Inspektor obiektów

Inspektor obiektów (ang. Object Inspector) umożliwia zmianę właściwości komponentów umieszczanych na formatce. Właściwości te również można zmieniać w kodzie programu. Wygląd Inspektora Obiektów ilustruje rysunek 1.3.1.

Rysunek 1.3.1

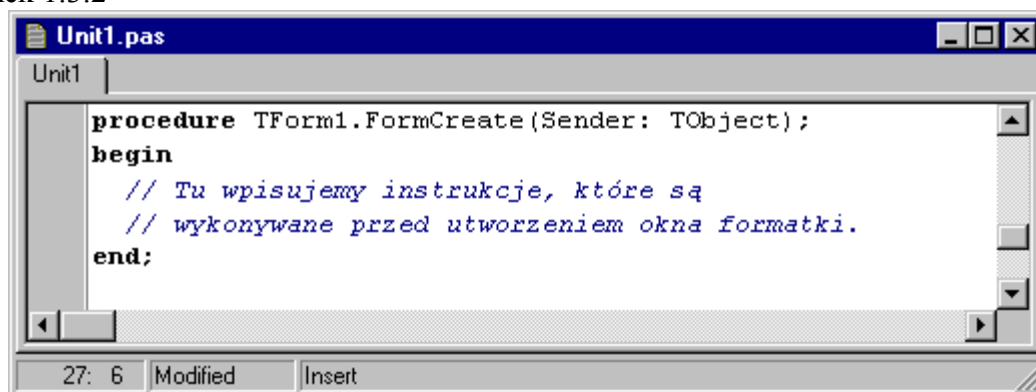


Po umieszczeniu komponentów na formatce możemy zmieniać ich właściwości za pomocą okna Inspektora Obiektów lub programowo. Sposób umieszczenia komponentów na formatce jest opisany w podrozdziale 1.1.

W celu wybrania zdarzenia (np. **OnCreate**) musimy wykonać następujące czynności:

- Kliknąć dwukrotnie na formatce (ang. Form), co spowoduje wygenerowanie zdarzenia (np. **OnCreate**). Rysunek 1.3.2 przedstawia taką sytuację:

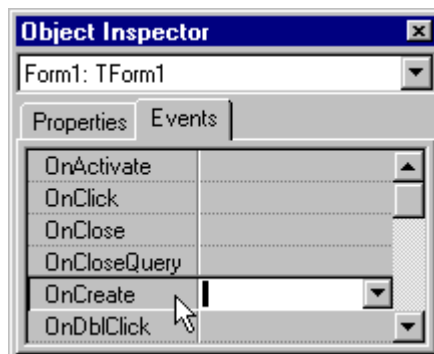
Rysunek 1.3.2



lub skorzystać z pomocy Inspektora Obiektów, z pomocą którego, możemy również wygenerować zdarzenie (np. **OnCreate**). Robimy to przez:

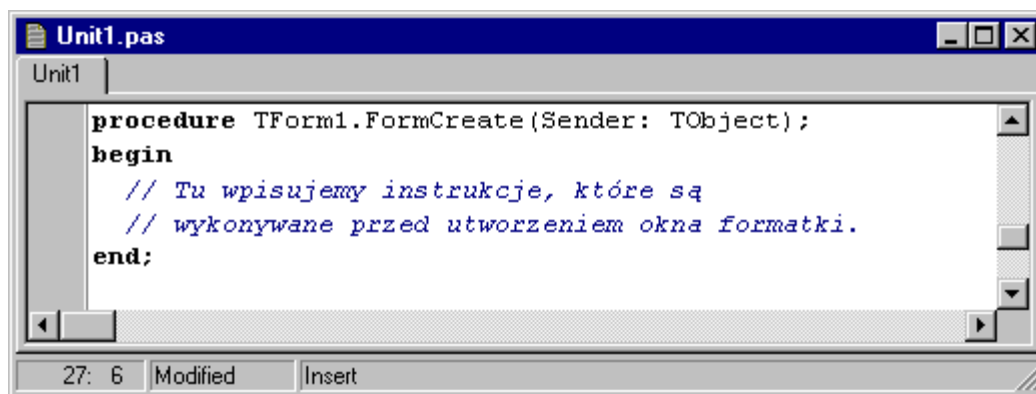
- Wybierz okno Inspektora Obiektów lub naciśnij klawisz funkcyjny F11;
- Następnie przejdź do zakładki **Events** (Zdarzenia) – rysunek 1.3.3;

Rysunek 1.3.3



- Obok okienka o nazwie (np. **OnCreate**) kliknij dwukrotnie, co spowoduje wygenerowanie zdarzenia – rysunek 1.3.4;

Rysunek 1.3.4

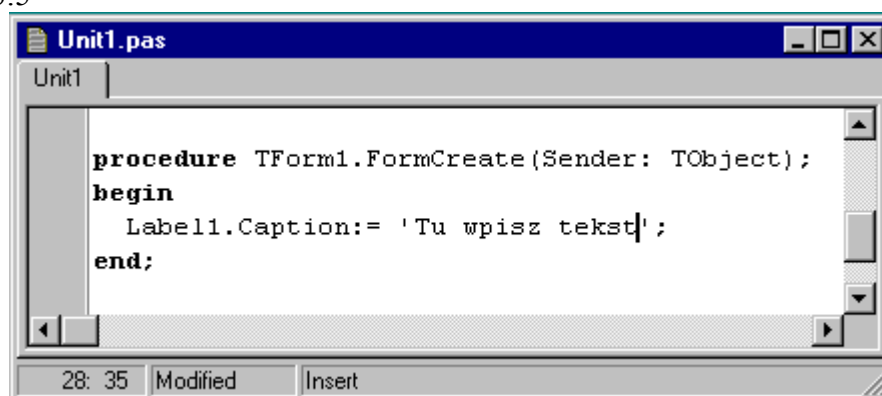


Przy wyborze innych zdarzeń wykonujemy takie same czynności jak opisane wyżej.

Zmianę właściwości (np. Caption) dokonujemy w następujący sposób:

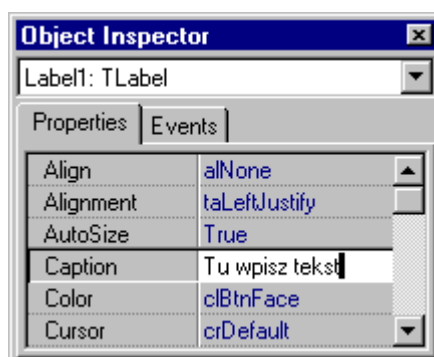
- Programowo, przez wpisanie w tym przypadku linii w kodzie programu – rysunek 1.3.5;

Rysunek 1.3.5



- Albo za pomocą okna **Object Inspector** (Inspektora Obiektów) - w tym przypadku komponent **Label** musi być zaznaczony -, przez wybranie właściwości (np. Caption) na zakładce **Properties** (Właściwości) i wpisanie dowolnego tekstu – rysunek 1.3.6;

Rysunek 1.3.6



Przy wyborze innych właściwości postępujemy tak samo.

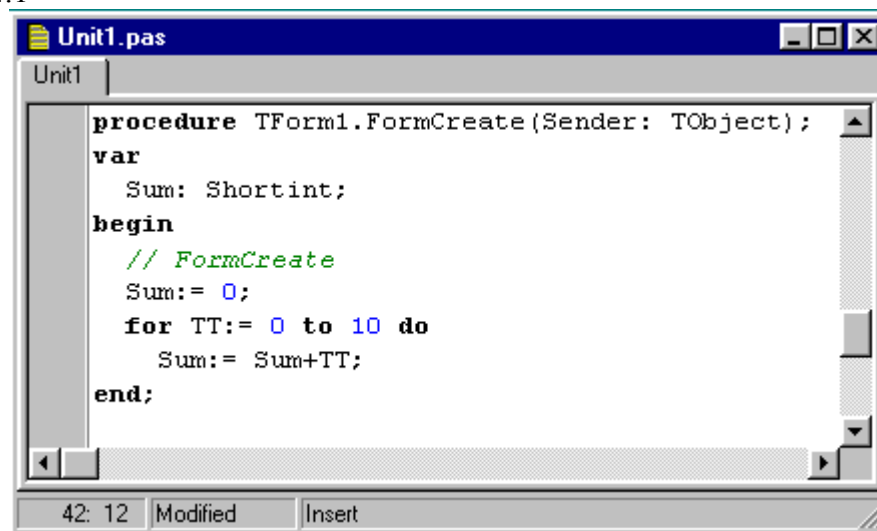
Uwaga:

- Właściwości i zdarzenia możemy zmieniać tylko komponentom znajdującym się na formacie za pomocą Inspektora Obiektów;
- Zawartość okna Inspektora Obiektów zmienia się w zależności od wybranego komponentu.

1.4. Edytor kodu

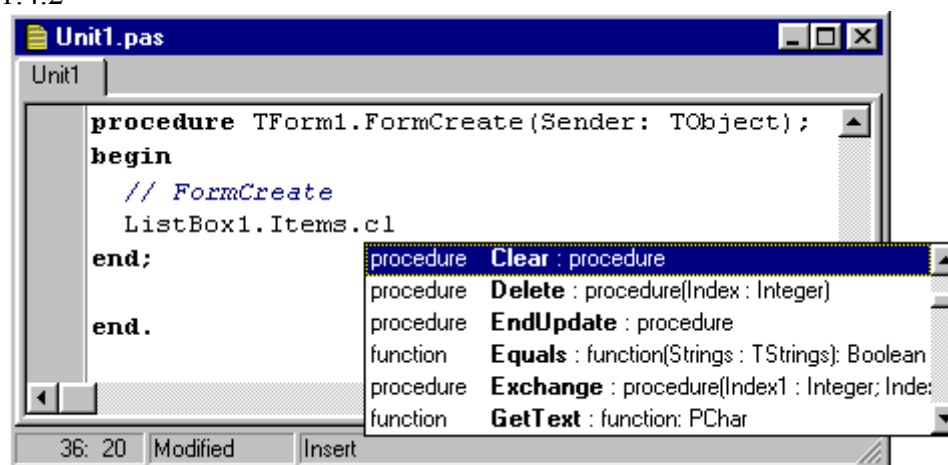
Edytor kodu jest edytorem tekstowym, w którym piszemy kod programu, który jest kolorowany dla większej przejrzystości. Kolory te możemy zmieniać w opcjach programu. Wygląd tego edytora przedstawia rysunek 1.4.1.

Rysunek 1.4.1



Edytor ten ma jeszcze trzy udogodnienia warte wspomnienia. Pierwsze udogodnienie polega na uzupełnianiu wpisywanych funkcji, przez co programista jest zwolniony od pamiętania właściwości poszczególnych komponentów. Taką sytuację przedstawia rysunek 1.4.2.

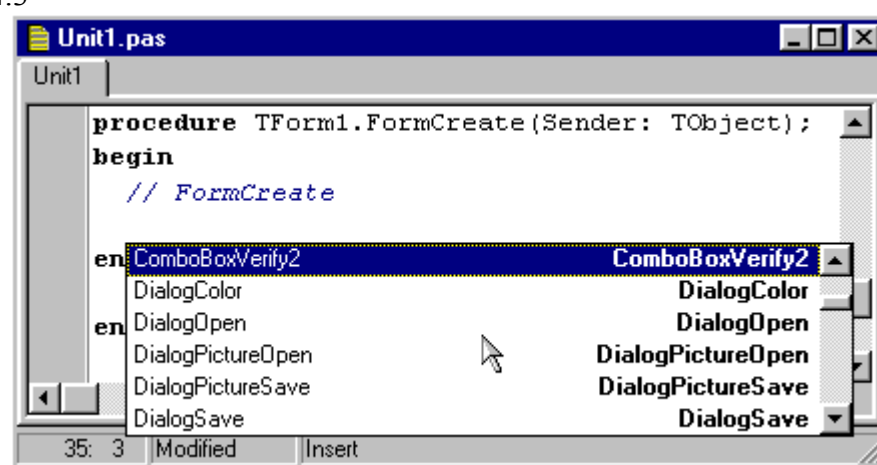
Rysunek 1.4.2



Drugim udogodnieniem jest możliwość wprowadzania fragmentu kodu do pisanej aplikacji za pomocą kombinacji klawiszy **CTRL+J**, co powoduje wyświetlenie listy dostępnych

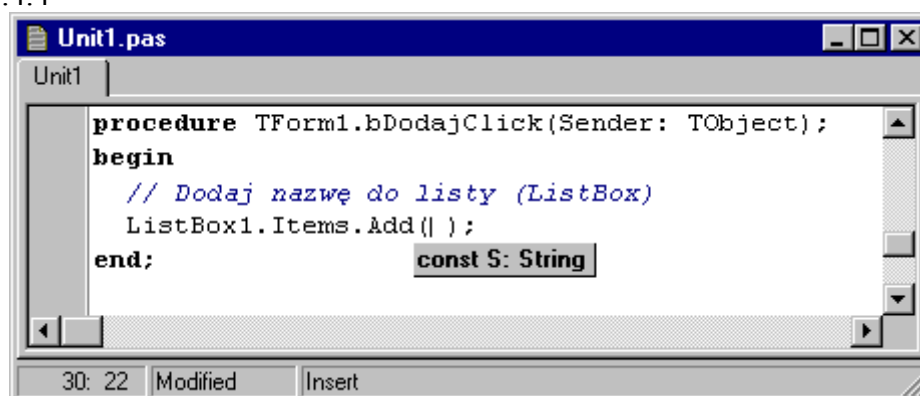
fragmentów kodu (wzorce kodu) - rysunek 1.4.3. Kody te musimy sobie wcześniej napisać i zapisać do pliku tekstowego DELPHI32.DCI, który znajduje się w głównym katalogu Delphi.

Rysunek 1.4.3



Trzecim udogodnieniem wartym wspomnienia jest podpowiadanie rodzaju zmiennej jaką należy wpisać (np. tekst w celu dodania do listy **ListBox**). Rysunek 1.4.4 przedstawia taką sytuację.

Rysunek 1.4.4



2. Typy plików w Delphi

Delphi rozróżnia następujące typy plików:

- DPR – plik projektu (zawiera dane na temat modułów i formatek, które wchodzi 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 nam opisanie fragmentów kodu. Komentarze te zwiększają czytelność kodu, co ma duże znaczenie dla programisty. 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  
}
```

- *// 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  
*)
```

3.2. Wybrane typy

W czasie deklaracji zmiennej należy określić jej typ. Typ jest zbiorem określającym wartości jakie może przyjmować zmienna.

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 2147483647 i ma rozmiar 4 bajtów;

Typy rzeczywiste:

- Real - przyjmuje liczby z zakresu od 2.9×10^{-39} do 1.7×10^{38} i ma rozmiar 6 bajtów (uwaga: typ ten został zachowany w celu zgodności z poprzednimi wersjami Pascala);
- Single - przyjmuje liczby z zakresu od 1.5×10^{-45} do 3.4×10^{38} i ma rozmiar 4 bajtów;
- Double - przyjmuje liczby z zakresu od 5.0×10^{-324} do 1.7×10^{308} i ma rozmiar 8 bajtów;
- Extended - przyjmuje liczby z zakresu od 3.4×10^{-4932} do 1.1×10^{4932} i ma rozmiar 10 bajtów;
- Comp - przyjmuje liczby z zakresu od $-263+1$ do $263-1$ 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;

Typ Variant:

Jest to dość niezwykle rozwiązanie pozwalające na dynamiczną zmianę typu zmiennej, tzn. zmienna może przyjąć typ całkowity, rzeczywisty, itp.

Przykład:

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

3.3. Zmienne

Zmienne umożliwiają przechowywanie wartości lub napis, gdy jest to potrzebne.

Zmienne znajdują się za słowem **var** (ang. variables) a przed blokiem instrukcji **begin** i **end** lub w sekcji **private** albo **public**.

Przykład:

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

Poniżej jest przedstawiony przykład prawidłowego umiejscowienia zmiennych w kodzie:

```
unit Unit1;
```

```
interface
```

```
uses
```

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

```
type
```

```
TForm1 = class(TForm)
  procedure FormCreate(Sender: TObject);
private
  { Private declarations }
  // Deklaracja zmiennej liczbowej typu całkowitego.
  // Zmienna globalna dla danego modułu
  numLiczba: Integer;
public
  { Public declarations }
  // Deklaracja zmiennej tekstowej
  // zmienna globalna dla całego projektu
  txtString: String;
end;
```

```
var
  Form1: TForm1;

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

implementation

```
{ $R *.DFM }
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  okSwitch: Boolean;
  // Deklaracja zmiennej logicznej
  // zmienna lokalna widoczna tylko w
  // funkcji, w której została zadeklarowana
begin
  // FormCreate
end;

end.
```

Zmiennej tekstowej o nazwie „txtString” (zadeklarowanej w formie nr 1, tj. Unit1 w sekcji **public**) możemy przypisać dowolny tekst z innej formy (np. nr 2, tj. Unit2). 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 programu. W przeciwnym przypadku musielibyśmy wprowadzać zmiany w kilku miejscach, co może przyczynić się do powstawania błędów. Stałą deklarujemy po słowie **const** (ang. constants). Oto przykłady:

const

```
Miesiące = 12; // Deklaracja ilości miesięcy w roku  
Doba = 24; // Deklaracja ilości godzin w ciągu doby
```

Poniżej jest przedstawiony przykład umiejscowienia stałych w kodzie:

```
unit Unit1;
```

interface**uses**

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

const

```
// Deklaracja stałej liczbowej  
// Stała globalna widoczna w całym programie  
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  
// Stała lokalna widoczna 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 savZapisz('nazwa');
```

Warunek ten powoduje zapisanie pliku, gdy zmienna 'okSave' będzie prawdziwa (czyli będzie miała wartość TRUE).

Przykład 2:

```
if (okSave = TRUE) then  
    savZapisz('nazwa') // Następuje zapisanie w momencie spełnienia warunku  
else  
    Close; // Następuje zamknięcie w przypadku spełnienia drugiego warunku
```

3.6. Instrukcja wiążąca With...Do

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

Przykład bez użycia instrukcji wiążącej **with**:

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

Przykład z użyciem instrukcji wiążącej **with**:

```
with Memo1 do  
begin  
    ReadOnly:= TRUE;  
    TabStop:= FALSE;  
    ScrollBars:= ssNone;  
    Color:= clBtnFace;  
    Lines.Clear;  
    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** powodująca 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;
```

- **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 znajduje się na dyskietce w katalogu Delphi\Inne\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 Delphi\Inne\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.

Konstrukcja:

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

Przykład znajduje się na dyskiecie w katalogu Delphi\Inne\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. Opis zdarzeń we właściwościach jest trochę inny niż wewnątrz kodu, np. **OnCreate** (na liście Inspektora Obiektu) jest równoznaczne z nazwą **FormCreate** (wewnątrz kodu) – odnosi się to do nazw wszystkich zdarzeń. Sposób generowania zdarzeń opisany jest w podrozdziale 1.3.

```
procedure TForm1.FormActivate(Sender: TObject);
begin
    // Tu wpisujemy instrukcje, które są wykonywane w momencie gdy formatka jest aktywna.
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 zapytanie użytkownika 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 rozmiarów 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 Delphi\Inne\Opis.

3.10. Nazwa aplikacji (programu)

Nazwa aplikacji, która jest wyświetlana na pasku zadań jest pobierana z nazwy pliku. W celu jej zmiany należy napisać **Application.Title:= 'Mój pierwszy program'**; pomiędzy słowami kluczowymi **begin** i **end**, w zdarzeniu **OnCreate** lub **OnShow**. Aby zdarzenie to zostało wygenerowane przez Delphi należy szybko dwukrotnie kliknąć na formatkę lub wybrać np. zdarzenie **OnCreate** przez dwukrotne szybkie kliknięcie w Inspektorze Obiektów (zakładka zdarzenia). W wygenerowanej procedurze, która jest wywoływana w momencie uruchamiania programu wpisujemy **Application.Title:= 'Mój pierwszy program'**; . Poniżej jest zamieszczony przykład.

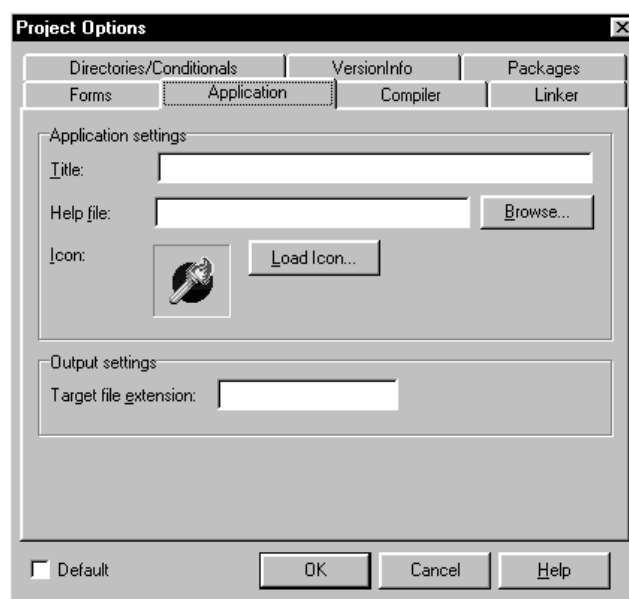
```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
  Application.Title:= 'Mój pierwszy program';  
end;
```

Od tej chwili na pasku zdań będzie się ukazywała nazwa „Mój pierwszy program”.

W celu zamiany ikony programu należy wykonać następujące czynności:

- Wybrać polecenie **Options** (Opcje) z menu **Project** (Projekt);
- W oknie **Project options** (Opcje projectu) należy wybrać zakładkę **Application** (Aplikacja) - rysunek 3.10.1;

Rysunek 3.10.1



- Wybrać klawisz z napisem **Load Icon** (Załaduj ikonę) i wybrać ikonę;
- Ostatni krok polega na zatwierdzeniu naszego wyboru przez wybranie klawisza OK.

3.11. Lista wybranych zmiennych globalnych

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

Do tych zmiennych wpisywane są wartości, pobrane z systemu w momencie uruchomienia programu. Wartości te można zmieniać, bez wpływu na zmiany ustawień w systemie. Zmienne te ułatwiają bardzo pracę programisty, zwalniając go ze żmudnego dostosowywania programu do różnych ustawień 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);
- **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);
- **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);
- **LongMonthNames** – Tablica znaków zawierająca pełne nazwy miesiąca (np. Caption:= LongMonthNames[2]; - zwróci nam pełną nazwę drugiego miesiąca);

- **ShortDayNames** – Tablica znaków zawierająca skrócone nazwy dnia (np. Caption:= ShortDayNames[1]; - zwróci nam skróconą nazwę pierwszego dnia);
- **LongDayNames** – Tablica znaków zawierająca pełne nazwy dnia (np. Caption:= LongDayNames[2]; - zwróci nam pełną nazwę drugiego dnia);

Przykład zastosowania wyżej wymienionych zmiennych:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    // FormCreate
    // Znak używany do oddzielenia roku, miesiąca, dnia.
    DateSeparator:= '-';

    // Znak używany 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;
```

3.12 Operatory logiczne

Operatory logiczne są wykorzystywane do testowania jednocześnie kilku warunków. Poniżej przedstawione są operatory wraz z przykładem ilustrującym jego 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 – Zmienna o nazwie „Liczba” ma przypisaną wartość 5.

Przykład równości:

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

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

Przykład nierówności:

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

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

Przykład większe lub równe:

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

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

Przykład zastosowania operatora AND:

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

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

Przykład zastosowania operatora OR:

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

Jeżeli będzie spełniony Warunek1 lub Warunek2 to wykonaj funkcję/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.

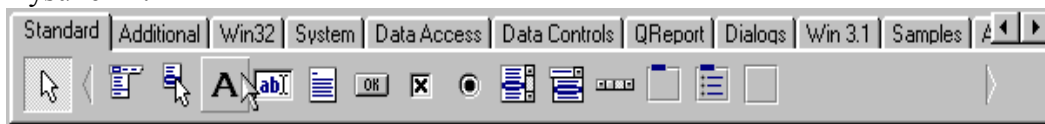
4. Pierwszy program

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 i wykonać następujące czynności:

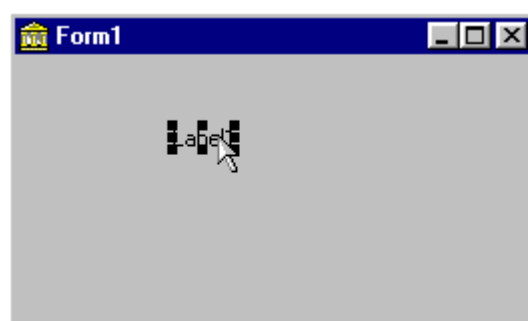
- Wybierz komponent **Label**  z palety komponentów **Standard** - rysunek 4.1;

Rysunek 4.1



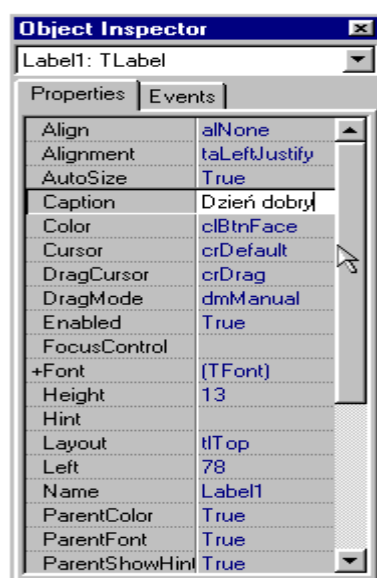
- Następnie komponent ten umieść na formacie - rysunek 4.2;

Rysunek 4.2



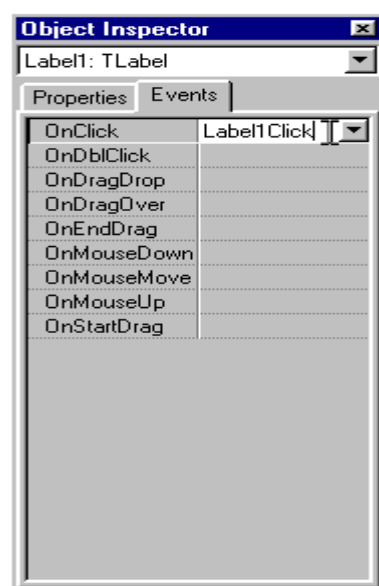
- W Inspektorze Obiektów wybierz właściwość **Caption** na zakładce **Properties** (Właściwości), w której wpisz słowa „Dzień dobry” - rysunek 4.3;

Rysunek 4.3



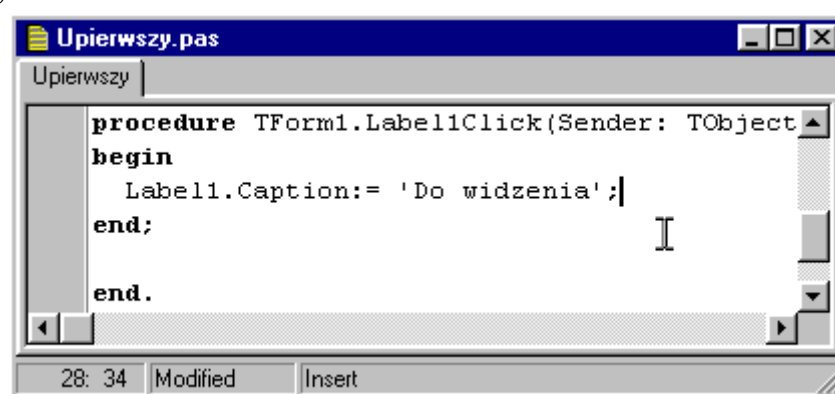
- Kolejnym krokiem będzie dwukrotne (szybkie) kliknięcie na komponencie **Label** znajdującym się na formacie lub wybranie zakładki **Events** (Zdarzenia) - **Object Inspector** (Inspektora obiektów) i kliknięcie na zdarzeniu **OnClick** – rysunek 4.4;

Rysunek 4.4



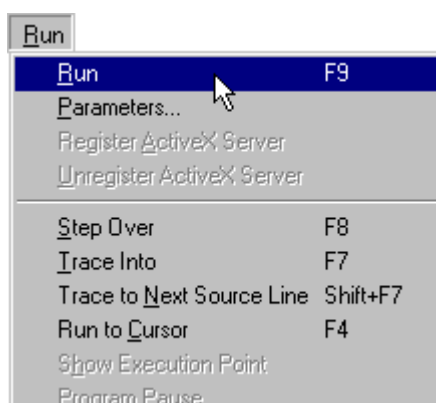
- W ostatnim kroku wpisz wiersz „Label1.Caption:= 'Do widzenia';”, w wygenerowanej przez program procedurze „Label1Click” – rysunek 4.5;

Rysunek 4.5



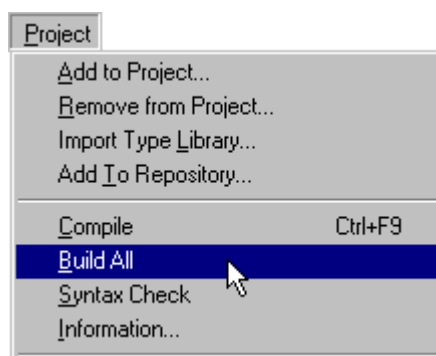
- Po tych krokach naciśnij klawisz F9 lub wybierz opcję **Run** (Uruchom) z menu **Run** w celu skompilowania i uruchomienia programu – rysunek 4.6.

Rysunek 4.6



Jeżeli chcesz uruchomić program poza środowiskiem Delphi to wystarczy go skompilować za pomocą opcji **Compile** (Kompiluj) lub **Build All** (Kompiluj wszystko) z menu **Project** (Projekt) – rysunek 4.7.

Rysunek 4.7



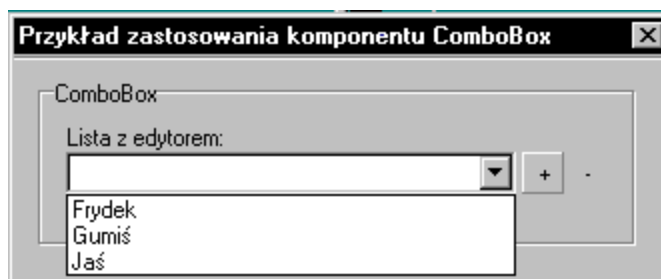
Opisany przykład znajduje się w katalogu Delphi\Pierwszy.

5. Ćwiczenia z podstawowych komponentów

Ćwiczenie 5.1. ComboBox

Stworzyć listę za pomocą komponentu **ComboBox** z możliwością dodania (bez możliwości dodania dwóch takich samych elementów) i usunięcia elementu z listy. Wygląd programu ilustruje rysunek 5.1.1.

Rysunek 5.1.1







Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\ComboBox.

Opis komponentu:



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

Sposób wykonania:

- Wybierz komponent **Label**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Następnie wybierz okno Inspektora Obiektów mając zaznaczony komponent **Label** lub naciśnij klawisz funkcyjny F11. Na zakładce **Properties** (Właściwości) wybierz właściwość **Caption** i wpisz „Lista z edytorem”;
- Wybierz komponent **ComboBox**  z palety komponentów (karta **Standard**);
- Wybierz komponent **SpeedButton**  z palety komponentów (karta **Additional**) i umieść go tuż z prawej strony komponentu **ComboBox**;
- Wybierz komponent **SpeedButton**  i umieść go tuż z prawej strony komponentu **SpeedButton**;
- Następnie mając zaznaczony komponent **SpeedButton** (ten bliżej komponentu **ComboBox**) wybierz okno Inspektora Obiektów i we właściwości **Caption** wpisz „+”. Potem wybierz właściwość **Hint** i wpisz „Dodaj do listy” i na koniec wybierz właściwość **ShowHint** i nadaj jej wartość TRUE (TRUE – pokazuje podpowiedzi);
- Z drugim komponentem **SpeedButton** postępuj tak samo jak z pierwszym, z tą różnicą, że we właściwości **Caption** wpisz „-”, oraz we właściwości **Hint** wpisz „Usuń z listy”;

- Kliknij dwukrotnie razy (szybko) na formatce, co spowoduje wygenerowanie zdarzenia **OnCreate**. W tym zdarzeniu wpisz 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;
```

W kolejnym kroku umieść deklarację tej funkcji (metody) 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 i między słowami **begin** i **end** wpisz kod obsługujący 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;

```

- Mając zaznaczony komponent **ComboBox**  wybierz okno Inspektora Obiektów i na zakładce **Events** (Zdarzenia) wybierz zdarzenie **OnKeyDown**, co spowoduje wygenerowanie procedury. Między słowami **begin** i **end** wygenerowanej procedury wpisz kod obsługi klawisza 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 z „+” spowoduje dodanie elementu do listy.

- Wybierz przycisk z napisem „-”, i kliknij na nim dwa razy (szybko), co spowoduje wygenerowanie procedury usuń. Między słowami **begin** i **end** wpisz kod odpowiedzialny za usuwanie elementów 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 ?',

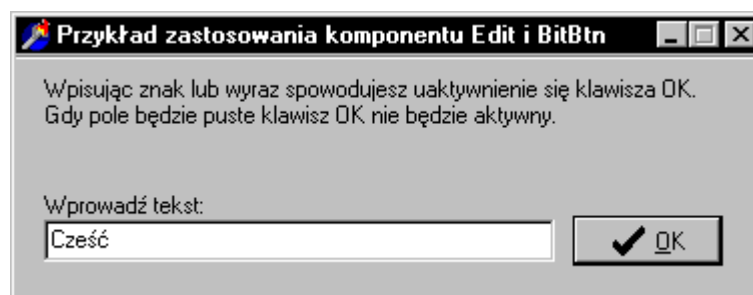
```

```
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;
```

Ćwiczenie 5.2. Edit, BitBtn

Napisz program, którego zadaniem jest uaktywnienie klawisza w momencie wprowadzenia tekstu. W innym przypadku klawisz ma być nieaktywny. Rysunek 5.2.1 przedstawia wygląd takiego programu.

Rysunek 5.2.1



Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\Edit.

Opis komponentów:





Edit to komponent, który znajduje się na karcie **Standard** palety komponentów. Umożliwia on wprowadzanie tekstu w stworzonej przez nas aplikacji.



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 A** (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (zakładka **Standard**);


- Mając zaznaczony ten komponent 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;
```

- Wybierz komponent **Edit**  znajdujący się na formatce i kliknij dwa razy (szybko) na nim, 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;
```

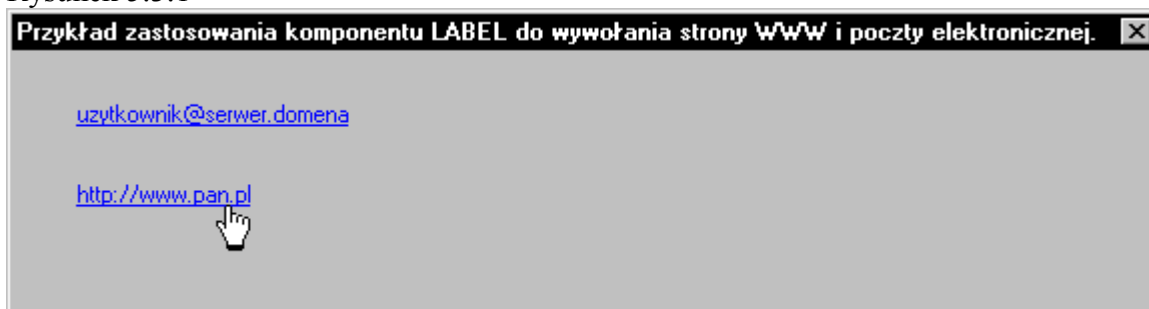
- Wybierz komponent **BitBtn**  znajdujący się na formatce i kliknij na nim dwa razy (szybko), co spowoduje wygenerowanie zdarzenia **OnClick**. W wygenerowanym zdarzeniu wpisz kod odpowiedzialny za wyświetlenie wprowadzonego tekstu do **Edit’a** na pasku tytułowym:

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

Ćwiczenie 5.3. Label

Napisz program wykorzystując komponent *Label* do uruchomienia przeglądarki internetowej i poczty elektronicznej. Rysunek 5.3.1 przedstawia wygląd programu.

Rysunek 5.3.1



Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\Label.

Opis komponentu:

A **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 komponenty **Label** **A** (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**) i umieść je na formacie jeden pod drugim;
- Kliknij dwa razy (szybko) na formacie 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 najechnia 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

// myszka 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 dwa razy (szybko) 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);

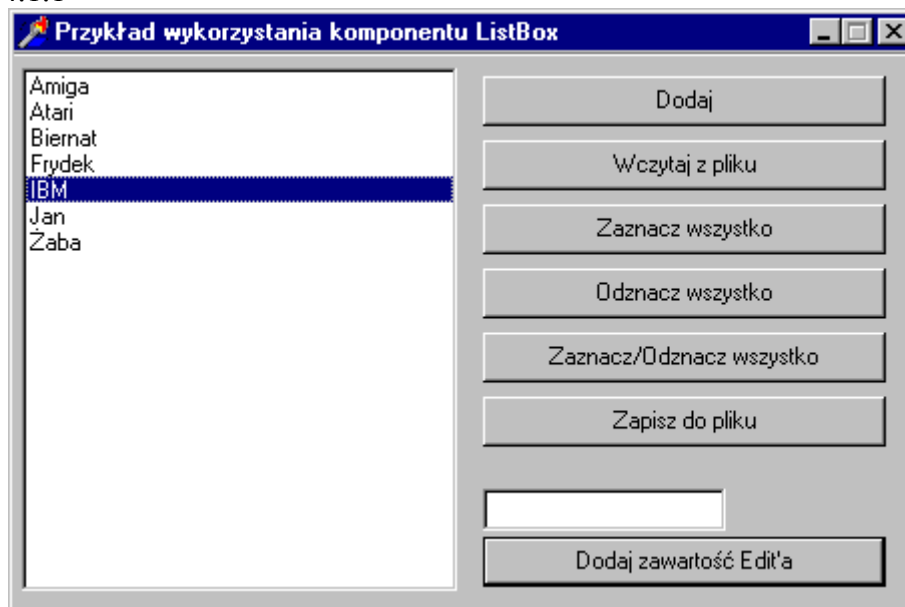
end;

Ćwiczenie 5.4. komponent ListBox

Ćwiczenie 5.4.1. ListBox

Napisz program, który będzie wczytywał i zapisywał listę, zaznaczał i odznaczał elementy listy oraz dodawał elementy do listy. Rysunek 5.4.1.1 przedstawia wygląd programu.

Rysunek 5.4.1.1






Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\ListBox\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**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponent **Edit**  (karta **Standard**);
- Wybierz kilka przycisków **Button**  (karta **Standard**) i opisać te przyciski zgodnie z rysunkiem 5.4.1.1;
- Kliknij dwa razy (szybko) na formatce i wpisz w wygenerowanym zdarzeniu **OnCreate** kod programu, który czyści zawartość listy i komponent **Edit**:

```
procedure TfrmMain.FormCreate(Sender: TObject);
```

beginListBox1.Items.Clear; *// Wyczyszczenie zawartości komponentu ListBox*Edit1.Text:= ""; *// Czyści zawartość komponentu Edit.***end;**

- Wybierz klawisz z napisem „Dodaj” i kliknij na nim dwa razy (szybko) w celu wygenerowania zdarzenia **OnClick** w którym wpiszesz 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 dwa razy (szybko) 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 dwa razy (szybko) na klawisz z napisem „Zaznacz/Odznacz wszystko” i w wygenerowanej procedurze wpisz kod:

```
procedure TfrmMain.bZaznaczOdznaczWszystkoClick(Sender: TObject);  
var  
    TT: Integer;  
begin  
    // Zaznacza/Odznacza wszystko  
    ListBox1.MultiSelect:= TRUE; // Włącza możliwość zaznaczania więcej niż jednej  
    pozycji.  
  
    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 dwa razy (szybko) na klawisz z napisem „Zapisz do pliku” i w wygenerowanej procedurze wpisz kod:

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

- Kliknij dwa razy (szybko) na klawisz z napisem „Zapisz do pliku” i w wygenerowanej procedurze 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 dwa razy (szybko) na klawisz z napisem „Odznacz wszystko” i w wygenerowanej procedurze 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 dwa razy (szybko) na klawisz z napisem „Zaznacz wszystko” i w wygenerowanej procedurze wpisz kod:

procedure TfrmMain.bZaznaczWszystkoClick(Sender: TObject);**var**

TT: Integer;

begin*// Zaznacza wszystko**// Włącza możliwość zaznaczania więcej niż jednej pozycji w komponencie ListBox*

ListBox1.MultiSelect:= TRUE;

*// Zaznaczenie wszystkich elementów***for** TT:= 0 **to** ListBox1.Items.Count-1 **do**

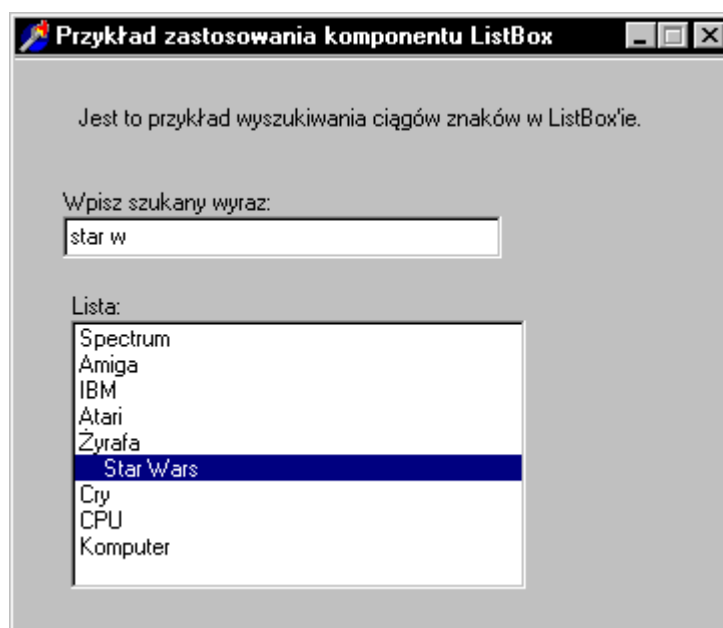
```

    ListBox1.Selected[TT]:= TRUE;
    // ListBox1.Selected[TT]:= TRUE; - Zaznacza pozycję o podanym numerze
end;
```

Ćwiczenie 5.4.2. ListBox



Wykonaj program, którego zadaniem jest znalezienie i zaznaczenie elementu w liście na podstawie pierwszych wprowadzonych liter w komponencie Edit. Wygląd tego programu przedstawiony jest na rysunku 5.4.2.1.

Rysunek 5.4.2.1



Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\ListBox\LBox2.

Sposób wykonania:

- Wybierz komponent **ListBox**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponent **Edit**  (karta **Standard**);
- Napisz funkcję wyszukującą elementy w liście. Przykładowa funkcja ma następującą postać:

```

function TfrmMain.AutomatyczneWyszukiwanie(txtSzukaj: String): Shortint;
var
    AA, TT, nL: Integer;
    txtFind: String;
begin
    // Automatyczne wyszukiwanie w ListBox'ie

    {
        Włącza możliwość zaznaczania więcej
```



```
niż jednej pozycji w komponencie ListBox
}
ListBox1.MultiSelect:= TRUE;

{
  Włącza stałą wielkość okna,
  co przyczynia się do widoczności w
  dolnej części okna całego elementu.
}
ListBox1.IntegralHeight:= TRUE;

// Odznacza wszystkie elementy
for AA:= 0 to ListBox1.Items.Count-1 do
  ListBox1.Selected[AA]:= FALSE;
  // Odznacza element o wybranym numerze, który znajduje się w zmiennej TT.

{
  Trim(' Jan ') - Likwiduje spacje po lewej i prawej stronie znaku.
  AnsiUpperCase('Frydek') - Zamienia cały wyraz na duże litery bez względu na język.
}
txtFind:= "";
txtFind:= AnsiUpperCase(Trim(txtSzukaj));

// Zwraca długość tekstu tzn. z ilu liter składa się wyraz lub zdanie.
nL:= 0;
nL:= Length(txtFind);

{
  Jeżeli zmienna 'txtFind' jest pusta to spełniony
  jest warunek po słowie ELSE, w przeciwnym wypadku
  spełniony jest warunek po słowie IF...THEN.
}
if (txtFind<>"") then
begin

  {
    Jeżeli pierwsze litery zgadzają się z
    literami zapisanymi w komponencie Edit, to
    zaznacz ten element w komponencie ListBox, w
    przeciwnym razie zaznacz pierwszy element na liście.
  }
  for TT:= 0 to ListBox1.Items.Count-1 do
  begin

    if (txtFind = Copy(AnsiUpperCase(Trim(ListBox1.Items[TT])), 1, nL)) then
    begin
      ListBox1.Selected[TT]:= TRUE;
      // zaznacza znaleziony element, którego numer znajduje się w zmiennej TT.

      Break; // Powoduje zakończenie działania i wyjście z pętli FOR.
```

```
end
else
begin
{
  Zaznacza pierwszy element na liście,
  jeżeli nie znaleziono żadnego wyrazu.
}
  ListBox1.Selected[0]:= TRUE;
end;
end;

end
else
begin
{
  Zaznacza pierwszy element na liście,
  jeżeli zmienna 'txtFind' jest pusta
}
  ListBox1.Selected[0]:= TRUE;
end;

{
  Wyłącza możliwość zaznaczania więcej
  niż jednej pozycji w komponencie ListBox
}
ListBox1.MultiSelect:= FALSE;
AutomatyczneWyszukiwanie:= 1; // Zwrócenie wyniku funkcji (1 – zadziałała).
end;
```

W kolejnym kroku umieść deklarację tej funkcji(metody) w typie obiektowym.

```
type
TfrmMain = class(TForm)
  function AutomatyczneWyszukiwanie(txtSzukaj: String): Shortint;
  procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```


➤ Kliknij dwa razy (szybko) na formatce i w wygenerowanej procedurze wpisz kod:

```
procedure TfrmMain.FormCreate(Sender: TObject);
begin
  Edit1.Text:= 'Tu wpisz szukane słowo'; // Wyświetlenie domyślnego tekstu.


  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');
end;
```

- W celu szybkiego przejścia z **Edit'a** do **ListBox'a** za pomocą strzałek góra/dół wybierz okno Inspektora Obiektów, mając zaznaczony komponent **Edit** . W zakładce **Events** (Zdarzenia) wybierz zdarzenie **OnKeyDown** i w wygenerowanej procedurze wpisz kod:

```
procedure TfrmMain.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
{
  Zdarzenie "onKeyDown" obsługuje naciśnięcie
  klawisza funkcyjnego, który powoduje przejście do ListBox'a.
  Tym klawiszem funkcyjnym jest "Strzałka w Dół".
}
  if (Key = VK_DOWN) or (Key = VK_UP) then
  begin
    // Powoduje uaktywnienie komponentu ListBox.
    ListBox1.SetFocus;
  end;
end;
```

- Kliknij dwa razy (szybko) na komponencie **Edit**  i w wygenerowanej procedurze wpisz kod, który wywoła funkcję do przeszukiwania listy:

```
procedure TfrmMain.Edit1Change(Sender: TObject);
begin
{
  Zdarzenie onChange powoduje automatyczne
  wykonywanie instrukcji(rozkazów) zawartych
  między słowami Begin...End.
  W tym przypadku jest to wywołanie funkcji przeszukującej listę.
}
  AutomatyczneWyszukiwanie(Edit1.Text); // Wywołanie funkcji wyszukującej.
end;
```

- Aby było możliwe przejście z komponentu **ListBox** do **Edit'a** przy naciśnięciu dowolnego klawisza alfanumerycznego musisz wpisać kod w wygenerowanym zdarzeniu **OnKeyPress**. Zdarzenie to możesz wygenerować wybierając - przy zaznaczonym komponencie **ListBox** - okno **Object Inspector** (Inspektora Obiektów).

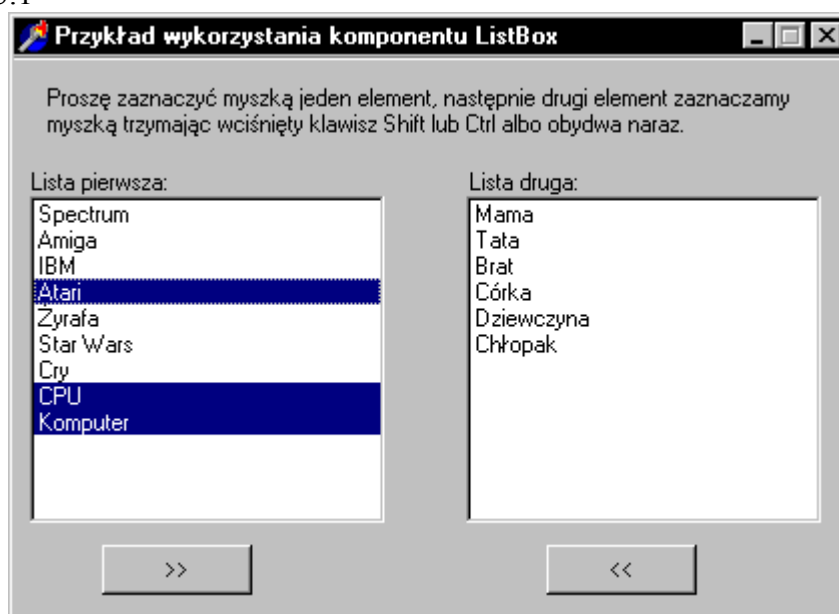
Wybierz zakładkę **Events** (Zdarzenia) i kliknij na zdarzeniu **OnKeyPress**, co spowoduje wygenerowanie procedury. W procedurze tej wpisz kod:

```
procedure TfrmMain.ListBox1KeyPress(Sender: TObject; var Key: Char);  
begin  
  {  
    Zdarzenie onKeyPress kontroluje naciśnięcie  
    dowolnego klawisza. Jeżeli nastąpi naciśnięcie  
    dowolnego klawisza, to zostaną wykonane instrukcje  
    zawarte między słowami Begin...End.  
  }  
  
  // Powoduje odznaczenie wypisanych znaków w komponencie Edit  
  Edit1.SetFocus; // Edit jest teraz aktywny.  
  Edit1.SelectAll; // Zaznaczenie tekstu.  
  Edit1.CopyToClipboard; // Skopiowanie do schowka.  
  Edit1.PasteFromClipboard; // Pobranie danych ze schowka.  
end;
```

Ćwiczenie 5.4.3. ListBox



Napisać program, w którym będzie możliwa zamiana zaznaczonych elementów między dwoma listami. Wygląd programu przedstawia rysunek 5.4.3.1.

Rysunek 5.4.3.1



Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\ListBox\LBox3.

Sposób wykonania:

- Wybierz 2 komponenty **ListBox**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**) i umieść je obok siebie;
- Wybierz 2 przyciski **Button**  (karta **Standard**) i opisać je zgodnie z rysunkiem 5.4.3.1;
- Kliknij dwa razy (szybko) 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 dwa razy (szybko) 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;
```

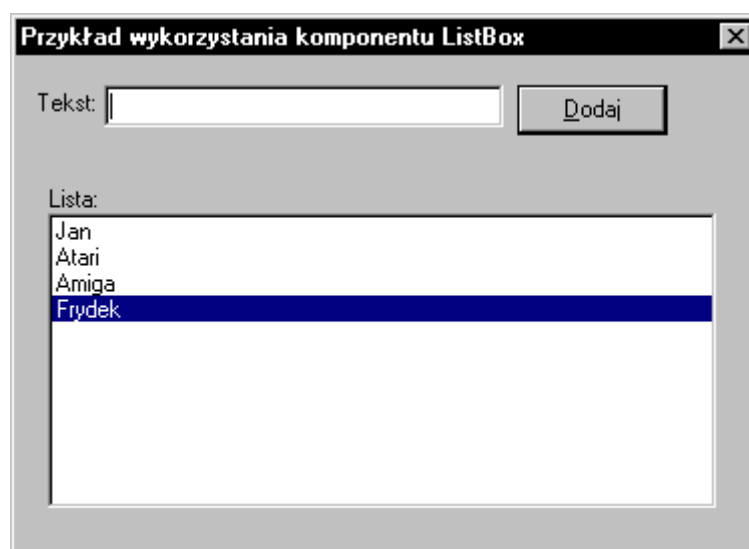
- Kliknij dwa razy (szybko) na klawiszu „<<” i w wygenerowanej procedurze wpisz kod:

```
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;
end;
```

Ćwiczenie 5.4.4. ListBox




Napisać program, który będzie zaznaczał dodany element do listy oraz nie będzie dopuszczał do dodania dwóch takich samych elementów. Rysunek 5.4.4.1 przedstawia wygląd programu.

Rysunek 5.4.4.1



Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\ListBox\LBox4.

Sposób wykonania:

- Wybierz komponent **ListBox**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponent **Edit**  (karta **Standard**);
- Wybierz przycisk **Button**  (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 umożliwi 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;
```

- W następnym kroku zadeklaruj napisane funkcje:

```

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

- Kliknij dwa razy (szybko) 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 dwa razy (szybko) 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;
```

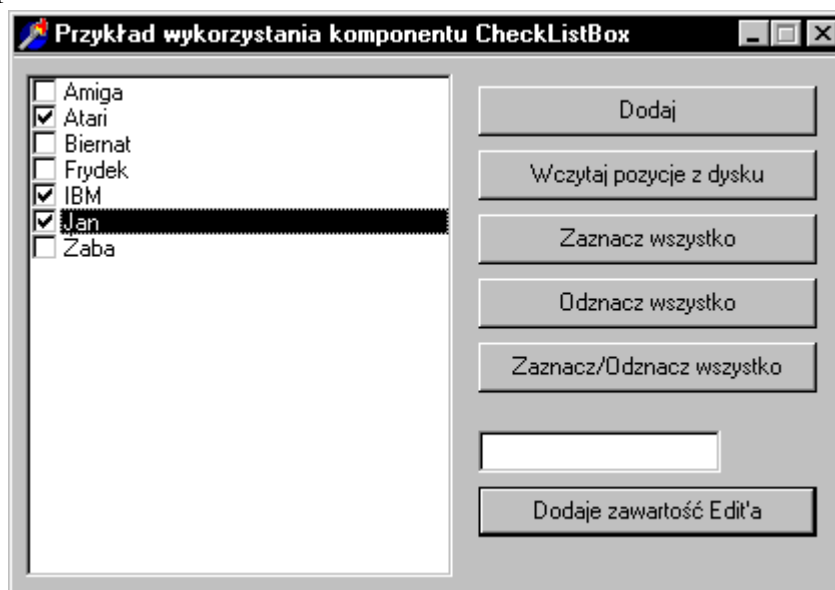
- Kliknij dwa razy (szybko) na klawisz z napisem „Dodaj” i w wygenerowanym zdarzeniu wpisz kod:

```
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;
```

Ćwiczenie 5.5. CheckListBox

Napisz program, który będzie wczytywał listę, zaznaczał i odznaczał elementy listy oraz dodawał elementy do listy. Rysunek 5.5.1 przedstawia wygląd programu.

Rysunek 5.5.1



Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\ListBox\LCBox.

Opis komponentu:



CheckListBox jest komponentem służącym do wyświetlania listy elementów z możliwością zaznaczania dowolnego elementu. Zaznaczanie elementu polega na postawieniu obok niego ptaszka. Znajduje się on na karcie **Additional** palety komponentów.

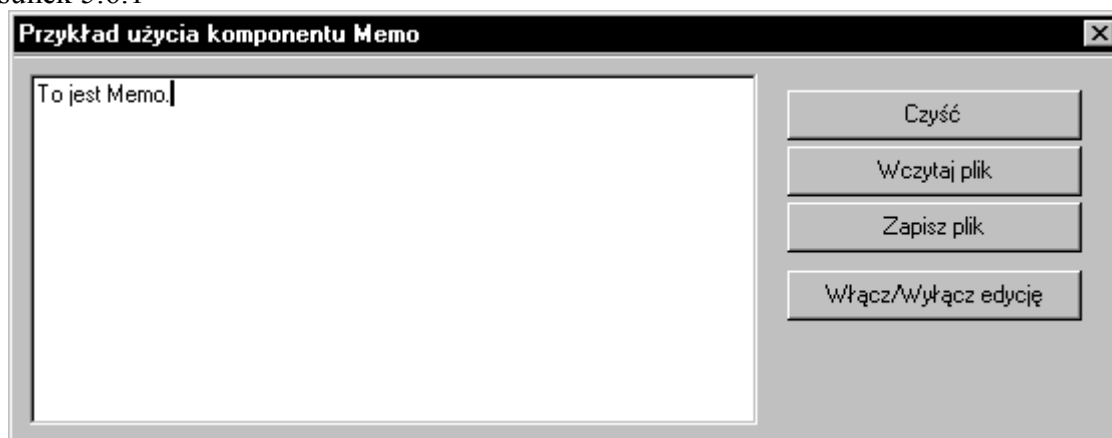
Sposób wykonania:

Rozwiązanie tego przykładu jest takie same jak w ćwiczeniu 5.4.1 z wyjątkiem nazwy komponentu, którą należy zmienić z nazwy **ListBox1** na nazwę **CheckListBox1**.

Ćwiczenie 5.6. Memo


Napisz program, którego zadaniem będzie wczytanie i zapisanie pliku, wyczyszczenie oraz włączenie i wyłączenie edycji. Rysunek 5.6.1 przedstawia wygląd programu.

Rysunek 5.6.1






Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\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**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponent **Edit**  (karta **Standard**);
- Wybierz kilka przycisków **Button**  (karta **Standard**) i opisać je zgodnie z rysunkiem 5.6.1;
- 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;
```

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

```
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;
```

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

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

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

```
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;
```

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

Ćwiczenie 5.7. Image

Napisz program, który będzie umożliwiał wczytanie dowolnego obrazka (bitmapy). Rysunek 5.7.1 przedstawia wygląd programu.

Rysunek 5.7.1





Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\Obraz.

Opis komponentu:



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

Sposób wykonania:

- Wybierz komponent Image  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Additional**);
- W oknie Inspektora Obiektów wybierz własność **Align** na karcie **Properties** (Własności) i nadaj jej wartość **alClient**, co spowoduje wykorzystanie przez komponent **Image** całej dostępnej przestrzeni formatki;
- Wybierz przycisk **Button**  (karta **Standard**) i opisz go zgodnie z rysunkiem 5.7.1;
- Wybierz przycisk i kliknij na nim dwa razy (szybko) oraz wpisz kod do wygenerowanej procedury:

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

```
    // Otwiera okno w celu wybrania pliku graficznego.
```

```
    OpenPictureDialog1.FileName:=""; // Brak nazwy pliku.
```

```
    OpenPictureDialog1.Execute; // Uruchomienie okna dialogowego.
```

```
if (Trim(OpenPictureDialog1.FileName)<>") then  
begin
```

```
    // Wczytanie pliku graficznego.
```

```
    Image1.Picture.LoadFromFile(Trim(OpenPictureDialog1.FileName));
```

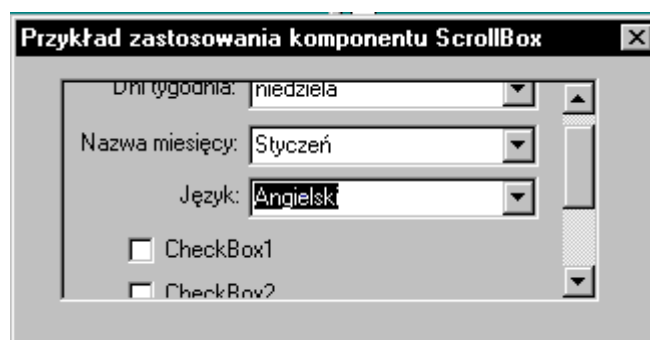
```
end;
```

```
end;
```

Ćwiczenie 5.8. ScrollBox

Napisz program demonstrujący działanie komponentu ScrollBox. Rysunek 5.8.1 przedstawia wygląd takiego programu.

Rysunek 5.8.1






Przykład znajduje się w katalogu Delphi\Cwicz\ScrlBox.

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 formacie. Komponent ten znajduje się na karcie **Additional** palety komponentów.

Sposób wykonania:


- Wybierz komponent **ScrollBox**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Additional**);
- Następnie wybierz kilkanaście dowolnych komponentów (np. **ComboBox** , **CheckBox**  itp.) i układaj je na komponencie **ScrollBox** tak, aby przesuwając komponent **ScrollBox**, inne komponenty przesuwały się razem z nim. Układać należy tak długo, aż całe miejsce będzie zajęte, wtedy to zobaczymy belkę przewijania;
- Przewiń obszar zajęty przez komponenty i na wolnym obszarze wstaw kilka nowych komponentów;
- Uruchom program i zobaczysz efekt.

Ćwiczenie 5.9. ProgressBar



Napisz program, który będzie wykonywał pętlę **for..to..do** i pokazywał postęp wykonywanej pracy.

Przykład znajduje się w katalogu Delphi\Cwicz\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.

Sposób wykonania:

- Wybierz komponent **ProgressBar**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Win32**);
- Następnie wybierz przycisk  (karta **Standard**) i kliknij na nim dwa razy (szybko) oraz w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.bDrugaInformacjaClick(Sender: TObject);
var
  TT: Integer;
begin
  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..To..Do.
  end;
```

Ćwiczenie 5.10. StatusBar




Napisz program taki jak w ćwiczeniu 5.9, który będzie informował o rozpoczęciu i zakończeniu działania pętli **for..to..do** wykorzystując do tego komponent **StatusBar**.

Przykład znajduje się w katalogu Delphi\Cwicz\StBar.

Opis komponentu:

 **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 we właściwości SimplePanel wybrać wartość TRUE w Inspektorze Obiektów. W innym przypadku jest możliwość podawania kilku informacji w jednym panelu równocześnie.

Sposób wykonania:

- Wybierz komponent **ProgressBar**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Win32**);
- Wybierz komponent **StatusBar**  z palety komponentów (karta **Win32**);
- Następnie wybierz przycisk  (Karta **Standard**) i kliknij na nim dwa razy (szybko) oraz w wygenerowanej procedurze wpisz kod:

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

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

Ćwiczenie 5.11. Timer


Napisz program, który będzie pokazywał bieżący czas.

Przykład znajduje się w katalogu Delphi\Cwicz\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 **Timer**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **System**);
- Kliknij na nim dwa razy (szybko) i w wygenerowanej procedurze **OnTimer** wpisz kod:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
```

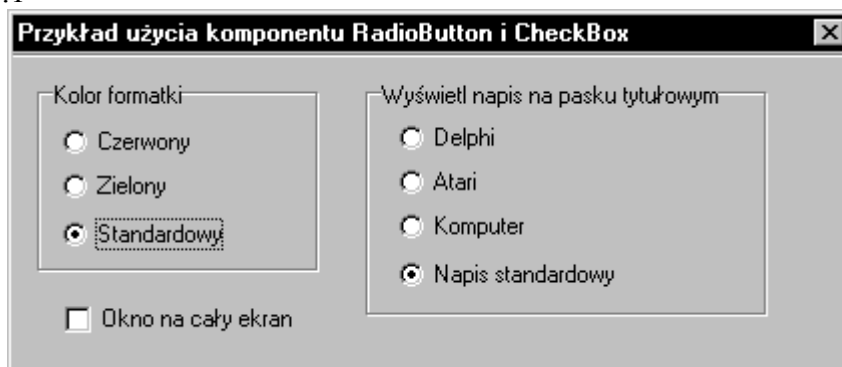


```
// Wyświetlenie godziny  
Label1.Caption:= TimeToStr(Time);  
// TimeToStr() - Dokonuje konwersji czasu na tekst  
end;
```

Ćwiczenie 5.12. RadioButton, GroupBox i CheckBox

Napisz program, który będzie zmieniał kolor formatki, napis na pasku tytułowym oraz powinien mieć możliwość maksymalizacji całego okna. Rysunek 5.12.1 przedstawia wygląd programu.

Rysunek 5.12.1



Przykład znajduje się w katalogu Delphi\Cwicz\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 wypełnione mniejszym czarnym kółeczkiem, w innym przypadku jest nie wypełnione.



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**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz kilka komponentów **RadioButton**  (karta **Standard**);
- Wybierz **CheckBox**  (karta **Standard**);
- Rozstaw i opisz wszystkie komponenty zgodnie z rysunkiem 5.12.1;
- Zaznacz pierwszy **RadioButton** w pierwszej ramce i kliknij na nim dwa razy (szybko). W wygenerowanej procedurze wpisz kod:

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

- Zaznacz pierwszy **RadioButton**  w drugiej ramce i kliknij na nim dwukrotnie (szybko). 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 Inspektora Obiektów wybierz zakładkę zdarzenia. Na zakładce tej wybierz zdarzenie **OnClick** i w liście rozwijanej wybierz procedurę **RadioButton4Click**. Tak postępuj z następnymi **RadioButton**'ami w ramce 2;
- Wybierz **CheckBox**  i kliknij na nim dwa razy (szybko) i w wygenerowanej procedurze wpisz kod:

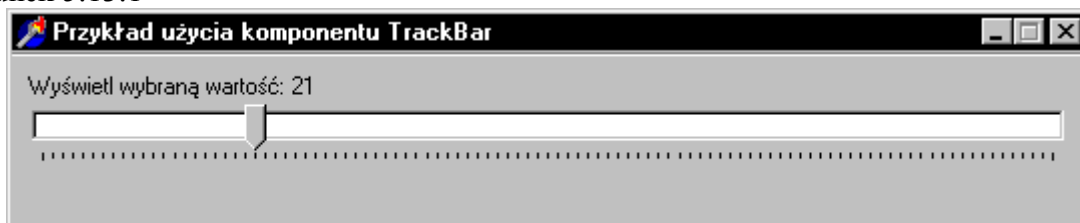
```
procedure TForm1.CheckBox1Click(Sender: TObject);  
begin  
  
    if (CheckBox1.Checked = FALSE) then  
        begin  
            Form1.WindowState:= wsNormal; // Przywraca poprzedni stan okna.  
        end  
    else  
        begin  
            Form1.WindowState:= wsMaximized; // Daje okno na cały ekran.  
        end  
end;
```

end;

Ćwiczenie 5.13. TrackBar

Wykonaj program pokazujący zmieniającą się wartość pod wpływem poruszania wskaźnikiem komponentu **TrackBar**. Rysunek 5.13.1 przedstawia program.

Rysunek 5.13.1




Przykład znajduje się w katalogu Delphi\Cwicz\TrackBar.

Opis komponentu:



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

Sposób wykonania:

- Wybierz komponent **Label A** (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponent **TrackBar**  (karta **Win32**);
- Zaznacz komponent **TrackBar** i w oknie Inspektora Obiektów we właściwości **Max** wpisz wartość 100;
- Mając zaznaczony komponent **TrackBar** kliknij na nim dwa razy (szybko) 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;
```

Ćwiczenie 5.14. MediaPlayer

Napisz program do odtwarzania plików dźwiękowych.



Przykład znajduje się w katalogu Delphi\Cwicz\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**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **System**);
- Wybierz komponent **Button**  (karta **Standard**) i kliknij na nim dwa razy (szybko). W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    // Otwórz plik muzyczny
    OpenFileDialog1.FileName:= "";
    OpenFileDialog1.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;
```

Ćwiczenie 5.15. Menu


Napisz program, w którym jest możliwość przełączania między dwoma Menu. Rysunek 5.15.1 przedstawia taki program.

Rysunek 5.15.1

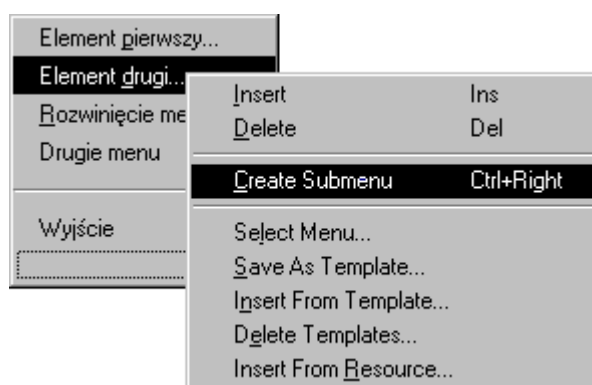


Przykład znajduje się w katalogu Delphi\Cwicz\Menu.

Opis komponentu:

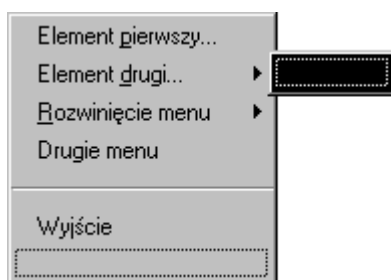
 **Menu** jest komponentem służącym do wywoływania funkcji programu uprzednio umieszczonych w menu. Na jednej formatce może być więcej niż jedno menu. Wybranie menu nas interesującego odbywa się za pomocą własności **Menu** obiektu formatki. Wyboru tego dokonujemy pisząc w programie linie **Menu:= MainMenu1**, 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**. Aby element posiadał podmenu musimy go wybrać i prawym klawiszem myszy wywołać menu podręczne, z którego należy wybrać funkcję **Create SubMenu** – rysunek 5.15.2.

Rysunek 5.15.2



Po wybraniu opcji **Create SubMenu** ukaże się nam puste podmenu – rysunek 5.15.3.

Rysunek 5.15.3

**Sposób wykonania:**

- Wybierz 2 komponenty **MainMenu** (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Pierwsze menu opisz zgodnie z rysunkiem 5.15.1;
- W drugim menu umieść funkcję **Menu pierwsze**, inne elementy menu są dowolne;
- Wybierz pierwszy element menu 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;
```

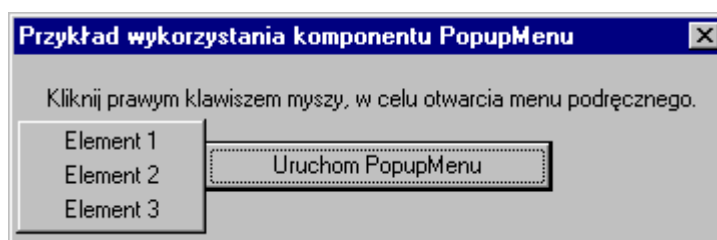
- Wybierz pozycję w menu drugim o nazwie **Menu pierwsze** i kliknij na nim. W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.MenuPierwsze1Click(Sender: TObject);  
begin  
    // Przełącza się na pierwsze menu  
    Menu:= MainMenu1;  
end;
```

Ćwiczenie 5.16. PopupMenu

Wykonaj program, w którym opcje będą wybierane za pomocą menu podręcznego. Rysunek 5.16.1 przedstawia taki program.

Rysunek 5.16.1





Przykład znajduje się w katalogu Delphi\Cwicz\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. Ponad to 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 2 komponenty **PopupMenu**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Opis elementów **PopupMenu** jest dowolny;
- Rozwiązanie i obsługa **PopupMenu** jest taka sama jak **MainMenu**;
- Wybierz klawisz **Button**  (karta **Standard**) i opisz go zgodnie z rysunkiem 5.16.1;
- Kliknij dwa razy (szybko) na formatce i w wygenerowanej procedurze wpisz kod:

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

- Kliknij na klawisz dwa razy (szybko) i w wygenerowanej procedurze wpisz kod:

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

```
begin
```

```
// Uruchomienie menu podręcznego
```

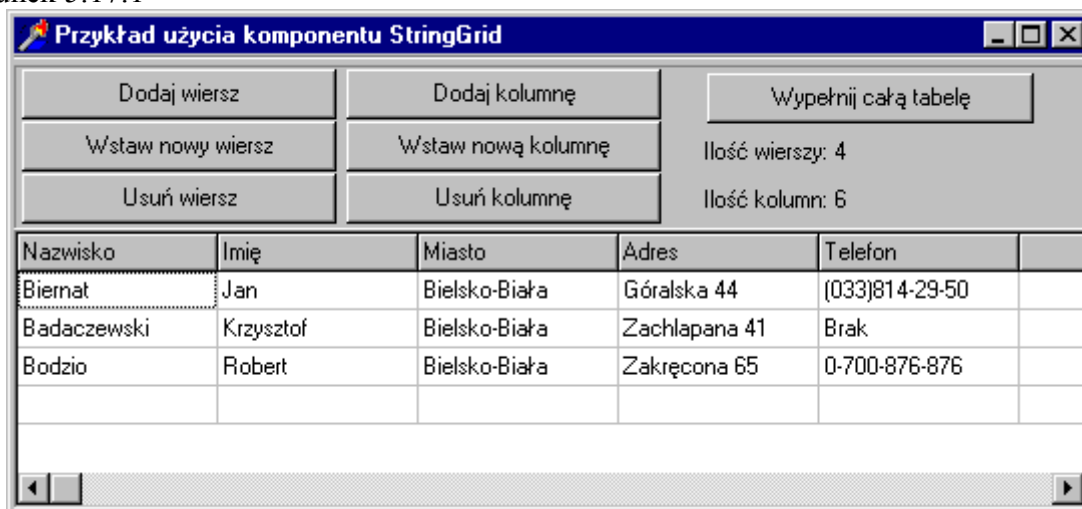
```
PopupMenu2.Popup(Left+Button1.Left+2, Top+Button1.Top-35);
```

```
end;
```

Ćwiczenie 5.17. StringGrid

Napisz program, który wypełni arkusz i będzie miał możliwość dodania wiersza lub kolumny oraz będzie miał możliwość wstawienia całego wiersza lub całej kolumny. Rysunek 5.17.1 przedstawia wygląd tego programu.

Rysunek 5.17.1





Przykład znajduje się w katalogu Delphi\Cwicz\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**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Additional**);
- Wybierz kilka klawiszy **Button**  (karta **Standard**) i opisz je zgodnie z rysunkiem 5.17.1;
- Wybierz klawisz z napisem „Dodaj wiersz” i kliknij dwa razy (szybko) na nim. W wygenerowanej procedurze wpisz kod:

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

```
begin
```

```
// Dodanie nowego wiersza
```

```
StringGrid1.RowCount:= StringGrid1.RowCount+1;
```


end;

- Wybierz klawisz z napisem „Wstaw nowy wiersz” i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem „Usuń wiersz” i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem „Dodaj kolumnę” i kliknij dwa razy (szybko) na nim. W wygenerowanej procedurze wpisz kod:

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

- Wybierz klawisz z napisem „Wstaw nową kolumnę” i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem „Usuń kolumnę” i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem „Wypełnij całą tabelę” i kliknij dwa razy (szybko) na nim. 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 dwa razy (szybko) na nim. W wygenerowanej procedurze wpisz kod:

```
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

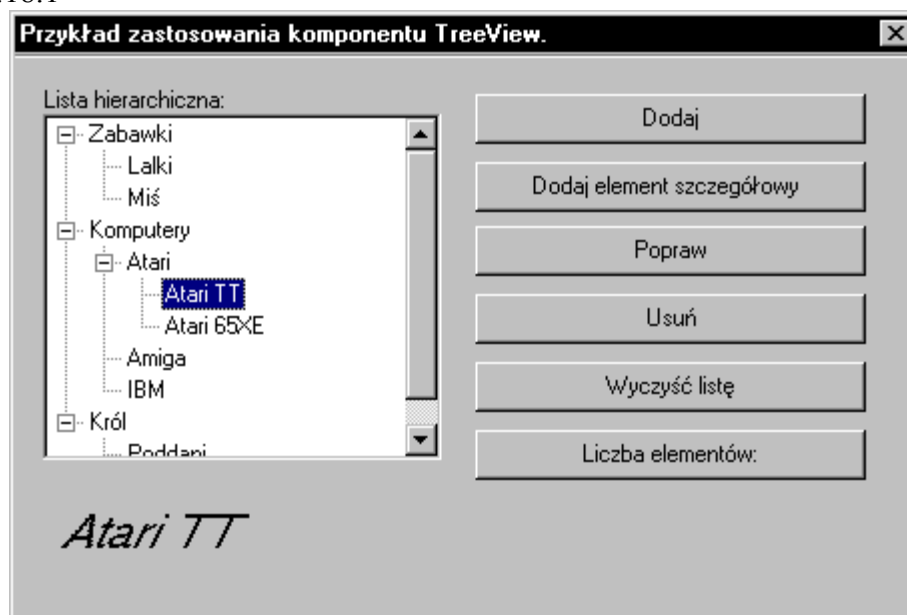
  // Zapisanie wypełnionych wierszy
  for aRow:= 0 to StringGrid1.RowCount-1 do
  for aCol:= 0 to StringGrid1.ColCount-1 do
  if (Trim(StringGrid1.Cells[numCol, aRow]) <> "") then
    Writeln(FT, StringGrid1.Cells[aCol, aRow]);

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

Ćwiczenie 5.18. TreeView

Napisz program, który przedstawia drzewo hierarchiczne pokazując przynależność poszczególnych elementów. Rysunek 5.18.1 przedstawia wygląd programu.

Rysunek 5.18.1



Przykład znajduje się w katalogu Delphi\Cwicz\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**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Win32**);
- Wybierz kilka klawiszy **Button**  (karta **Standard**) i opisz je zgodnie z rysunkiem 5.18.1;
- Wybierz komponent **Label**  (karta **Standard**) i umieść go pod komponentem **TreeView**;
- Wybierz klawisz z napisem „Dodaj” i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem „Dodaj element szczegółowy” i kliknij dwa razy (szybko) na nim. 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 'Element1'.
        Przez co tworzy się hierarchia.
    }
end;
```

```
// InputBox('Dodaj', 'Tekst:', 'Pozycja') - otwiera okno edycyjne  
end;
```

- Wybierz klawisz z napisem „Popraw” i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem „Usuń” i kliknij dwa razy (szybko) na nim. W wygenerowanej procedurze wpisz kod:

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

- Wybierz klawisz z napisem „Wyczyść listę” i kliknij dwa razy (szybko) na nim. W wygenerowanej procedurze wpisz kod:

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

- Wybierz klawisz z napisem „Liczba elementów” i kliknij dwa razy (szybko) na nim. 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 A** nazwy zaznaczonego elementu podczas poruszania się po liście trzeba wybrać komponent **TreeView**, gdy będzie na formatce i kliknąć na nim dwa razy (szybko). W wygenerowanej procedurze wpisać następujący 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;

Ćwiczenie 5.19. 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 Delphi\Inne\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 pola **Private** (Prywatnego) deklaracji „List: TStringList;”. Przykład wstawienia deklaracji jest przedstawiony poniżej.

private

{ Private declarations }

List: TStringList; *// Obiekt reprezentujący naszą listę.*


- Następnie kliknij dwa razy (szybko) na formatce, co spowoduje wygenerowanie zdarzenia **OnCreate**. W wygenerowanej procedurze twórz listę przez wpisanie linii „List:= TStringList.Create;”. Przykład jest zamieszczony poniżej.

procedure TForm1.FormCreate(Sender: TObject);

begin

```
List:= TStringList.Create; // Dynamiczne tworzenie listy
```

end;

- Wybierz klawisz **Button**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Kliknij dwa razy (szybko) 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;

- Ostatnim krokiem będzie usunięcie listy z pamięci, co robimy umieszczając linię „List.Destroy;” w metodzie obsługującej zdarzenie **OnClose**. Wstawienie tej linii ilustruje poniższy przykład.

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
```

begin

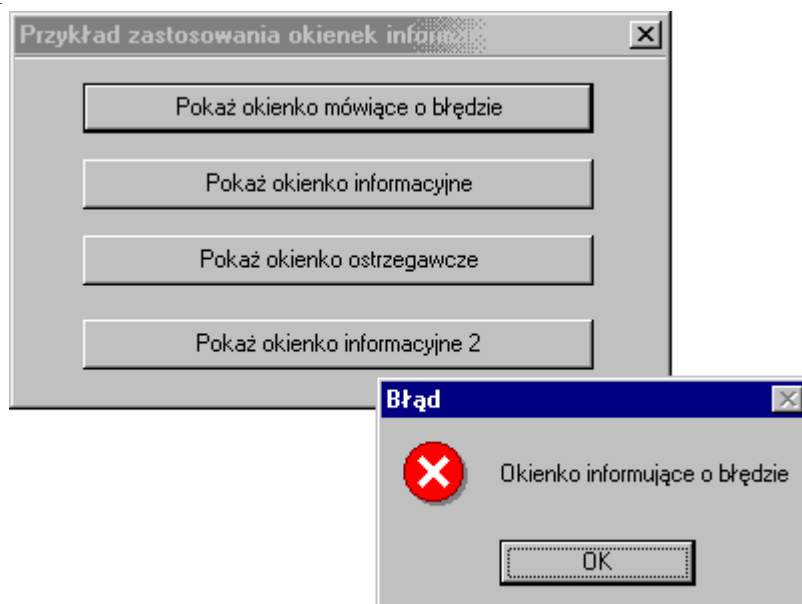
```
List.Destroy; // Usunięcie listy z pamięci
```

end;

Ćwiczenie 5.20. Okna informacyjne

Napisz program, który będzie prezentował rodzaje okien informacyjnych. Rysunek 5.20.1 przedstawia taki program.

Rysunek 5.20.1



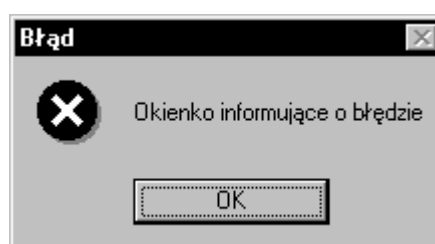
Przedstawiony przykład znajduje się w katalogu Delphi\Inne\OknaInf.

Opis okien informacyjnych:

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

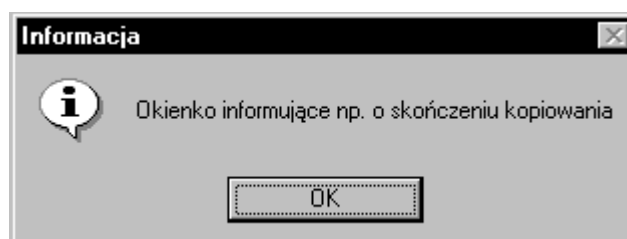
- Informujące o błędzie – rysunek 5.20.2;

Rysunek 5.20.2



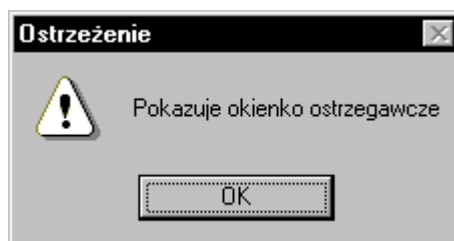
- Informacyjne – rysunek 5.20.3;


Rysunek 5.20.3



- Ostrzegawcze – rysunek 5.20.4;

Rysunek 5.20.4

**Sposób wykonania:**

- Wybierz kilka klawiszy **Button**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**) i opisz je zgodnie z rysunkiem 5.20.1;
- Wybierz klawisz z napisem **Pokaż okno mówiące o błędzie** i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem **Pokaż okno informacyjne** i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem **Pokaż okno ostrzegawcze** i kliknij dwa razy (szybko) na nim. 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;

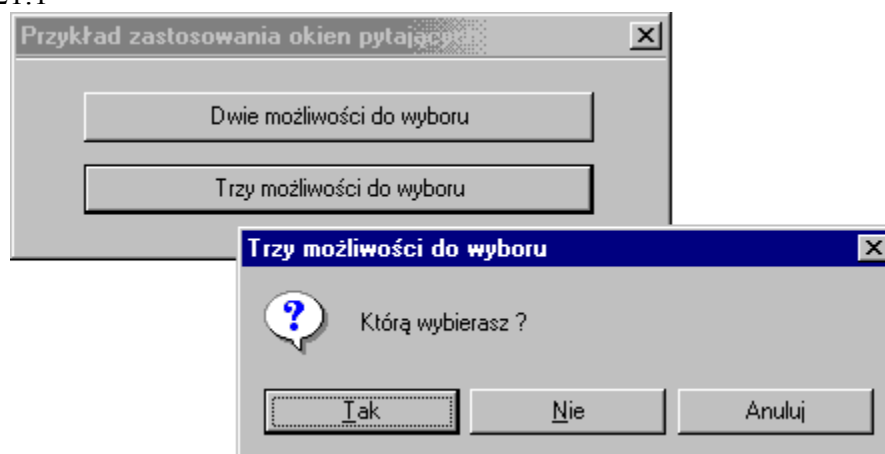
- Wybierz klawisz z napisem **Pokaż okno informacyjne 2** i kliknij dwa razy (szybko) na nim. W wygenerowanej procedurze wpisz kod:

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

Ćwiczenie 5.21. Okna służące do zadawania pytań

Napisz program, który będzie prezentował okna służące do zadawania pytań. Rysunek 5.21.1 przedstawia taki program.

Rysunek 5.21.1



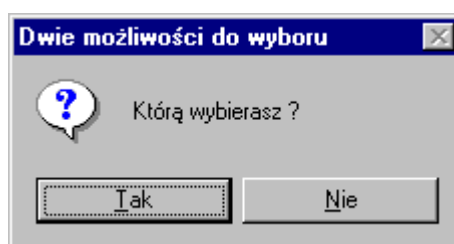
Przedstawiony przykład znajduje się w katalogu Delphi\Inne\OknaPyt.

Opis okien służących do zadawania pytań:

Okna pytające służą do zadawania pytań w trakcie działania aplikacji. Rozróżnia się następujące rodzaje okien pytających:

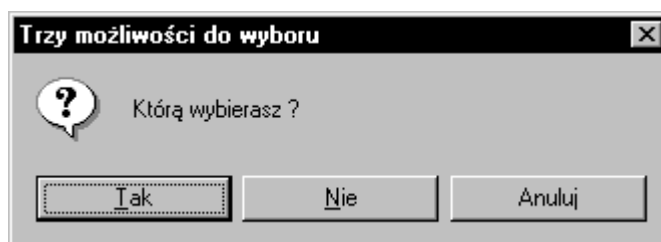
- Dwoma możliwymi odpowiedziami – rysunek 5.21.2;


Rysunek 5.21.2



- Trzema możliwymi odpowiedziami – rysunek 5.21.3;

Rysunek 5.21.3

**Sposób wykonania:**

- Wybierz dwa klawisze **Button**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**) i opisz je zgodnie z rysunkiem 5.21.1;
- Wybierz klawisz z napisem **Dwie możliwości do wyboru** i kliknij dwa razy (szybko) na nim. W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  numBtn: Integer;
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;
```

- Wybierz klawisz z napisem **Trzy możliwości do wyboru** i kliknij dwa razy (szybko) na nim. W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  numBtn: Integer;
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;

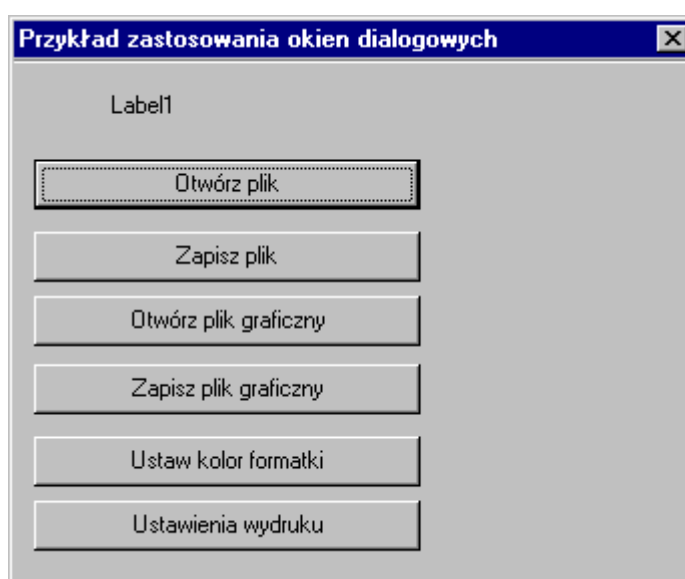
if (numBtn = IDNO) then
begin
  ShowMessage('To jest DRUGA możliwość');
end;

if (numBtn = IDCANCEL) then
begin
  ShowMessage('To jest TRZECIA możliwość');
end;
end;
```

Ćwiczenie 5.22. Okna dialogowe

Napisz program, który będzie prezentował okna dialogowe (np. okno służące do zapisywania plików). Rysunek 5.22.1 przedstawia wygląd programu.

Rysunek 5.22.1



Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\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 klawiszy **Button**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**) i opisz je zgodnie z rysunkiem 5.22.1;
- Wybierz następujące komponenty z karty **Dialogs**: - **Open Dialog** (otwórz plik) ,
- **Save Dialog** (zapisz plik) , - **Open Picture Dialog** (otwórz plik graficzny) ,
- **Save Picture Dialog** (zapisz plik graficzny) , - **Color Dialog** (okno kolorów) ,
- **Printer Setup Dialog** (ustawienia wydruku) ;
- Wybierz klawisz z napisem „Otwórz plik” i kliknij dwa razy (szybko) na nim. 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;

```

- Wybierz klawisz z napisem „Zapisz plik” i kliknij dwa razy (szybko) na nim. 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;

```

- Wybierz klawisz z napisem „Otwórz plik graficzny” i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem „Zapisz plik graficzny” i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem „Ustaw kolor formatki” i kliknij dwa razy (szybko) na nim. 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;
```

- Wybierz klawisz z napisem „Ustawienia wydruku” i kliknij dwa razy (szybko) na nim. W wygenerowanej procedurze wpisz kod:

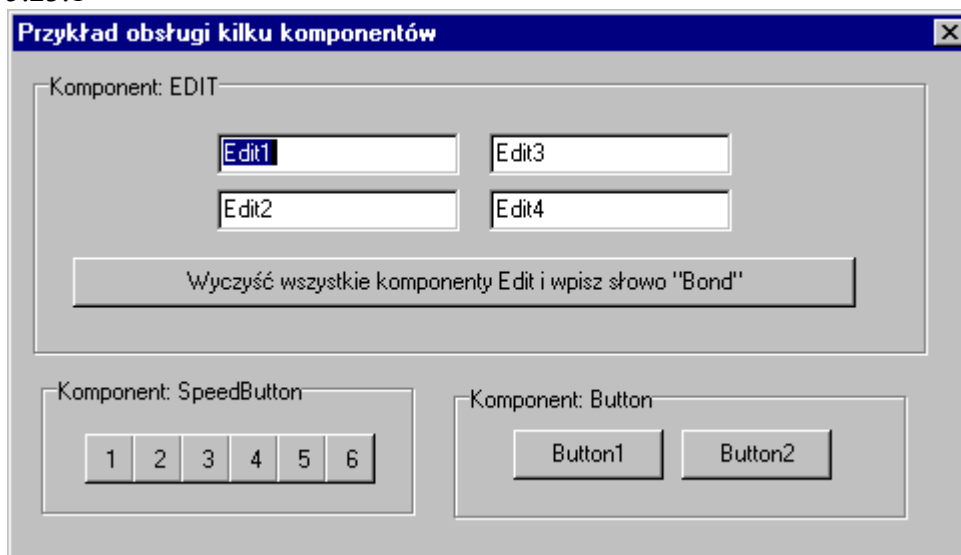
```
procedure TForm1.bUstawieniaWydrukuClick(Sender: TObject);  
begin  
  // Otwiera okienko z ustawieniami wydruku  
  PrinterSetupDialog1.Execute;
```

end;

Ćwiczenie 5.23. Obsługa kilku komponentów

Wykonaj program, który będzie obsługiwał kilka komponentów za pomocą jednej funkcji.
Rysunek 5.23.1 przedstawia wygląd tego programu.

Rysunek 5.23.1






Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\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 obsługiwać 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**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**) i rozmieść je zgodnie z rysunkiem 5.23.1;
- Wybierz kilka komponentów **Edit**  (karta **Standard**) oraz jeden komponent **Button**  (karta **Standard**) i umieść je w pierwszym GroupBox'ie według rysunku 5.23.1;
- Wybierz ten przycisk i kliknij na niego dwa razy (szybko) 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 zgodnie z rysunkiem 5.23.1;
- Kliknij na pierwszy klawisz **SpeedButton** dwa razy (szybko) 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;

```

- Wybierz drugi przycisk i we właściwości **OnClick** zakładki Zdarzenia w oknie Inspektora Obiektów w liście rozwijanej, wybierz nazwę procedury obsługującej pierwszy klawisz (w naszym przykładzie jest to procedura SpeedButton1Click). Postępuj tak z resztą przycisków **SpeedButton**.
- Wybierz dwa klawisze **Button**  (karta **Standard**) i umieść je w trzecim GroupBox'ie tak jak na rysunku 5.23.1;
- Wybierz pierwszy klawisz **Button** i kliknij dwa razy (szybko). W wygenerowanej procedurze wpisz kod:

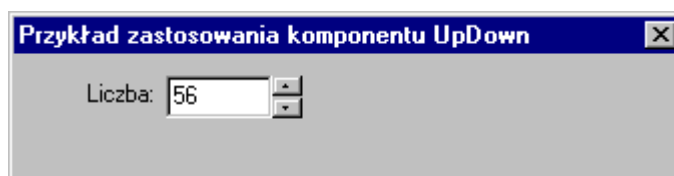
```
procedure TForm1.Button1Click(Sender: TObject);
begin
    // Obsługa kilku przycisków za pomocą jednej funkcji.
    if ((Sender as TButton).Tag = 1) then ShowMessage((Sender as TButton).Caption);
    if ((Sender as TButton).Tag = 2) then ShowMessage((Sender as TButton).Caption);
    {
        Tag – jest to numer kolejnego komponentu, który ustawia się we właściwościach.
        Za pomocą tego numeru można łatwo zidentyfikować komponent.
    }
end;
```

- Wybierz drugi przycisk i we właściwości **OnClick** zakładki Zdarzenia okna Inspektora Obiektów w liście rozwijanej wybierz nazwę procedury obsługującej pierwszy klawisz (w naszym przykładzie jest to procedura Button1Click).

Ćwiczenie 5.24. UpDown

Wykonaj program, który będzie zwiększał lub zmniejszał liczbę wyświetlaną w komponencie Edit. Rysunek 5.24.1 przedstawia wygląd takiego programu.

Rysunek 5.24.1



Przykład znajduje się w katalogu Delphi\Cwicz\UpDown.

Opis komponentu:





UpDown umożliwia zwiększanie lub zmniejszanie liczby wyświetlanej za pomocą komponentu np. Edit. We właściwości Associate komponentu UpDown znajdują się listy komponentów, które mogą być z nim powiązane.

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 komponent **UpDown**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Win32**);

- Wybierz komponent **Edit**  (karta **Standard**);
- Wybierz właściwość **Associate** komponentu **UpDown** będąc w oknie Inspektora Obiektów i z listy rozwijanej wybierz **Edit1**. Wybór ten spowoduje połączenie komponentu **Edit** z komponentem **UpDown**, który zostanie przysunięty do komponentu **Edit**.

Ćwiczenie 5.25. ScrollBar


Wykonaj program, który będzie przewijał rysunek w ograniczonym polu. Rysunek 5.25.1 przedstawia wygląd programu.

Rysunek 5.25.1






Przykład znajduje się w katalogu Delphi\Cwicz\ScrBar.

Opis komponentu:

 **ScrollBar (Belka przewijania)** jest komponentem, który służy do przewijania np. obrazu wyświetlanego na ograniczonym obszarze.

Sposób wykonania:

- Wybierz komponent **Panel1**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponent **Image**  (karta **Additional**) i postaw go na komponent **Panel1** tak, aby przesuważąc komponent **Panel1**, komponent **Image1** przesuwał się razem z nim;
- Wybierz komponent **ScrollBar**  (karta **Standard**) i ustaw go pionowo (Właściwość **Kind** = **sbVertical**) obok komponentu **Panel1**;
- Wybierz komponent **ScrollBar**  (karta **Standard**) i ustaw go poziomo (Właściwość **Kind** = **sbHorizontal**) obok komponentu **Panel1**;
- Kliknij dwa razy (szybko) na formatce i w wygenerowanej procedurze wpisz kod:

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

```
// Wczytanie obrazu do komponentu Image.
```

```
Image1.Picture.LoadFromFile('las.bmp');
```

```
// Przypisanie do właściwości Max komponentu ScrollBar faktycznej wielkości obrazka.
```

```
ScrollBar1.Max:= Image1.Picture.Bitmap.Height-Panel1.Height;
```

```
ScrollBar2.Max:= Image1.Picture.Bitmap.Width-Panel1.Width;
```

```
end;
```

- Wybierz komponent **ScrollBar** położony pionowo i kliknij na niego dwa razy (szybko). W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
```

```
begin
```

```
  // Przewijanie obrazka w dół i do góry.
```

```
  Image1.Top:= (ScrollBar1.Position*-1);
```

```
end;
```

- Wybierz komponent **ScrollBar** położony poziomo i kliknij na niego dwa razy (szybko). W wygenerowanej procedurze wpisz kod:

```
procedure TForm1.ScrollBar2Change(Sender: TObject);
```

```
begin
```

```
  // Przewijanie obrazka w prawo i w lewo.
```

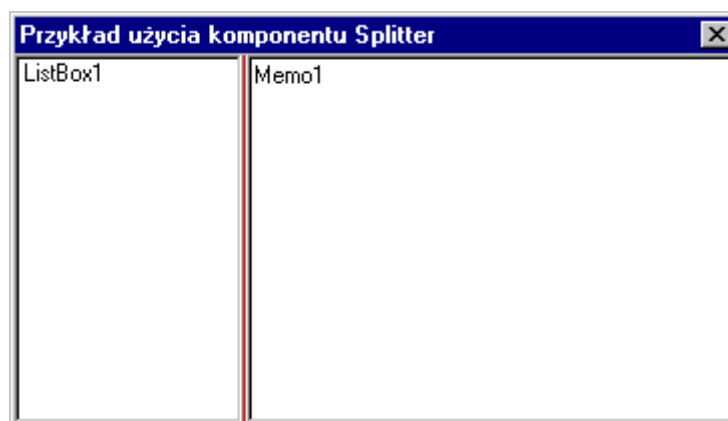
```
  Image1.Left:= (ScrollBar2.Position*-1);
```

```
end;
```

Ćwiczenie 5.26. Splitter

Wykonaj program, który będzie umożliwiał zmianę dwóch obszarów znajdujących się na jednej formacie. Na rysunku 5.26.1 przedstawiony jest taki program.

Rysunek 5.26.1






Przykład znajduje się w katalogu Delphi\Cwicz\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**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**) i we właściwości **Align** wybierz 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**.

Ćwiczenie 5.27. Dynamiczne tworzenie komponentów

Wykonaj program, który wczyta 10 ikon do komponentów *Image*. Komponenty te mają być tworzone w momencie wczytywania ikon oraz ma być możliwość zapisania na dysk wczytanych rysunków. Rysunek 5.27.1 przedstawia ten program.

Rysunek 5.27.1





Przykład znajduje się w katalogu Delphi\Cwicz\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 **ScrollBar**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Additional**);
- Wybierz komponent **ListBox**  (karta **Standard**);
- Ułóż wyżej wybrane komponenty zgodnie z rysunkiem 5.27.1;
- Napisz funkcję wczytującą nazwy plików o rozszerzeniu „ico” do **ListBox’a**. Postać takiego kodu może wyglądać następująco:

```
function TForm1.ListBoxSzukajPlikow(txtPath, txtExt: String): Shortint;
var
  SR : TSearchRec;
begin
  {
    Wczytanie do listy nazwy plików.
    Jeżeli funkcja nie znajdzie plików o odpowiednim
    rozszerzeniu to zwróci wartość -1.
    W przeciwnym przypadku zwróci wartość 1.
  }
  ListBoxSzukajPlikow:= -1;

  ListBox1.Items.Clear; // Wyczyszczenie zawartości listy.

  {
    Włączenie automatycznego dopasowania wysokości.
    Umożliwia widoczność ostatniego elementu listy.
  }
  ListBox1.IntegralHeight:= FALSE;

  ListBox1.Sorted:= TRUE; // Włączenie sortowania.

  {
    Sprawdzenie czy w katalogu istnieją pliki o podanym rozszerzeniu.
    Funkcję wykonuje się tak długo, aż wszystkie pliki spełniające
    warunek zostaną wczytane do ListBox'a.
  }
  if (FindFirst(Trim(txtPath+txtExt), faAnyFile, SR) = 0) then
  begin
    repeat
      if (SR.Attr<>faDirectory) then
      begin
        ListBox1.Items.Add(Trim(SR.Name)); // Dodanie elementu
        ListBoxSzukajPlikow:= 1; // Funkcja zwraca 1, gdy dodany będzie element
      end;
    until (FindNext(SR)<>0);
    FindClose(SR);
  end;
end;

Zadeklaruj funkcję przez wpisanie jej w sekcji opisowej (interface);
```

```

type
  TForm1 = class(TForm)
    function ListBoxSzukajPlikow(txtPath, txtExt: String): Shortint;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

```

- Zadeklaruj obiekty **Panel** i **Image** jako tablice. Postać deklaracji wygląda następująco:

```

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    dccPanel: array[1..100] of TPanel; // Deklaracja obiektu typu TPanel
    dccImage: array[1..100] of TImage; // Deklaracja obiektu typu TImage
  public
    { Public declarations }
  end;

```

- Napisz funkcję zapisującą rysunki na dysk:

```

procedure TForm1.dccZapiszJako(Sender: TObject);
begin
  // Zapisuje plik pod inną nazwą

  // Wstawienie znaku pustego jako domyślnej nazwy pliku.
  SavePictureDialog1.FileName:="";

  // Uruchomienie okna dialogowego do zapisu rysunku.
  SavePictureDialog1.Execute;

  {
    Sprawdzenie, czy została podana nazwa pliku.
    Jeżeli tak, to dokonaj zapisu, w przeciwnym przypadku nie wykonuj nic.
  }
  if (Trim(SavePictureDialog1.FileName) <> "") then
    begin

      // Zapisanie pliku na dysk.
      dccImage[(Sender as
        TImage).Tag].Picture.SaveToFile(Trim(SavePictureDialog1.FileName));
      {
        (Sender as TImage).Tag - konstrukcja ta umożliwia
        przekazanie własności, które są wspólne dla wszystkich
        komponentów Image. W tym przykładzie chodzi o

```

*przekazanie identyfikatora wybranego komponentu Image.
Sender - jest referencją do obiektu, którego dotyczy zdarzenie
(np. kliknięcie myszką)*

```
}
end;
end;
```

Zadeklaruj funkcję przez wpisanie jej w sekcji opisowej (**interface**);

```
type
  TForm1 = class(TForm)
    procedure dccZapiszJako(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

➤ Kliknij na formie dwa razy (szybko) i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  TT, numY: Integer; // Deklaracja zmiennych
begin
  // Wyszukanie dostępnych ikon w bieżącym katalogu.
  ListBoxSzukajPlikow(".", '*.ico');

  // Przypisanie zmiennej 'numY' wartości zero.
  numY:= 0;

  // Dynamiczne tworzenie komponentu.
  for TT:= 0 to ListBox1.Items.Count-1 do
  begin

    //-- Panel --
    // Utworzenie obiektu typu TPanel
    dccPanel[1+TT]:= TPanel.Create(ScrollBar1);
    {
      dccPanel[1+TT]:= TPanel.Create(ScrollBar1);
      Wszystkie komponenty posiadają konstruktor Create, który
      jest wywoływany z jednym parametrem. Parametr ten określa
      właściciela tworzonego obiektu. Najczęściej właścicielem
      tworzonego obiektu jest obiekt formatki (czyli TForm1), na
      którym to ma się znaleźć tworzony obiekt.
    }

    // Ustalenie pozycji Y
    dccPanel[1+TT].Top:= 9+numY;
```



```
// Ustalenie pozycji X
dccPanel[1+TT].Left:= 9;

// Ustalenie szerokości
dccPanel[1+TT].Width:= 49;

// Ustalenie wysokości
dccPanel[1+TT].Height:= 49;

// Wyczyszczenie wyświetlanego tekstu
dccPanel[1+TT].Caption:= "";

// Umieszczenie stworzonego obiektu na np. formacie
ScrollBar1.InsertControl(dccPanel[1+TT]);

//-- Image --
// Utworzenie obiektu typu TImage
dccImage[1+TT]:= TImage.Create(dccPanel[1+TT]);

// Ustalenie pozycji Y
dccImage[1+TT].Top:= 2;

// Ustalenie pozycji X
dccImage[1+TT].Left:= 3;

// Ustalenie szerokości
dccImage[1+TT].Width:= dccPanel[1+TT].Width-5;

// Ustalenie wysokości
dccImage[1+TT].Height:= dccPanel[1+TT].Height-5;

// Wyłączenie przezroczystości komponentu Image.
dccImage[1+TT].Transparent:= FALSE;

{
  Wyłączenie możliwości dopasowywania się
  komponentu Image do wielkości rysunku.
}
dccImage[1+TT].AutoSize:= FALSE;

// Wyłączenie możliwości rozciągnięcia rysunku.
dccImage[1+TT].Stretch:= FALSE;

// Umieszczenie obrazka w środku
dccImage[1+TT].Center:= TRUE;

// Wczytanie obrazka
dccImage[1+TT].Picture.LoadFromFile(ListBox1.Items[TT]);

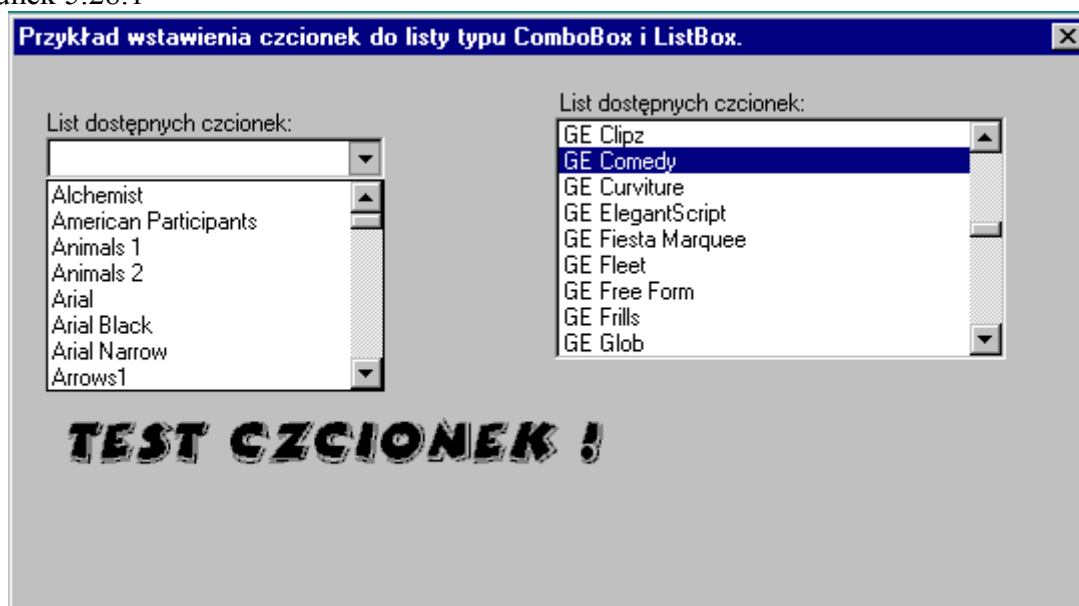
// Przypisanie metody "dccZapiszJako" do zdarzenia OnClick.
```

```
dccImage[1+TT].OnClick:= dccZapiszJako;  
  
// Nadanie identyfikatora tworzonemu obiektom.  
dccImage[1+TT].Tag:= 1+TT;  
  
{  
  Włączenie możliwości wyświetlania podpowiedzi w  
  momencie najechania kursorem myszy na dany obiekt.  
}  
dccImage[1+TT].ShowHint:= TRUE;  
  
// Wstawienie do podpowiedzi nazwy plików.  
dccImage[1+TT].Hint:= ListBox1.Items[TT];  
  
// Umieszczenie stworzonego obiektu na np. formatce  
dccPanel[1+TT].InsertControl(dccImage[1+TT]);  
  
{  
  Licznik umożliwiający wstawienie tworzonych  
  obiektów z góry do dołu o pewną wielkość  
  (np. wysokość tworzonych obiektów).  
}  
numY:= numY+55;  
end;  
end;
```

Ćwiczenie 5.28. Wczytanie czcionek do listy ComboBox i ListBox




Wykonaj program, który wczyta dostępne w systemie czcionki do listy **ComboBox** i **ListBox**. Rysunek 5.28.1. przedstawia wygląd takiego programu.

Rysunek 5.28.1



Przykład znajduje się w katalogu Delphi\Cwicz\Fonty.

Sposób wykonania:

- Wybierz komponent **ComboBox**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponent **ListBox**  (karta **Standard**);
- Wybierz trzy komponenty **Label**  (karta **Standard**) i dwa z nich opisz według rysunku 5.28.1;
- Ułóż wyżej wybrane komponenty zgodnie z rysunkiem 5.28.1;
- Kliknij na formę dwa razy (szybko) 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;
```

- Zaznacz komponent **ComboBox**  i kliknij na nim dwa razy (szybko) i w wygenerowanej procedurze wpisz kod:


```
procedure TForm1.ComboBox1Change(Sender: TObject);
```

```
begin
```

```
  // Pokaż czcionkę po jej wybraniu.
```

```
  Label3.Font.Name:= ComboBox1.Text;
```

```
end;
```

- Zaznacz komponent **ListBox**  i kliknij na nim dwa razy (szybko) 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;
```

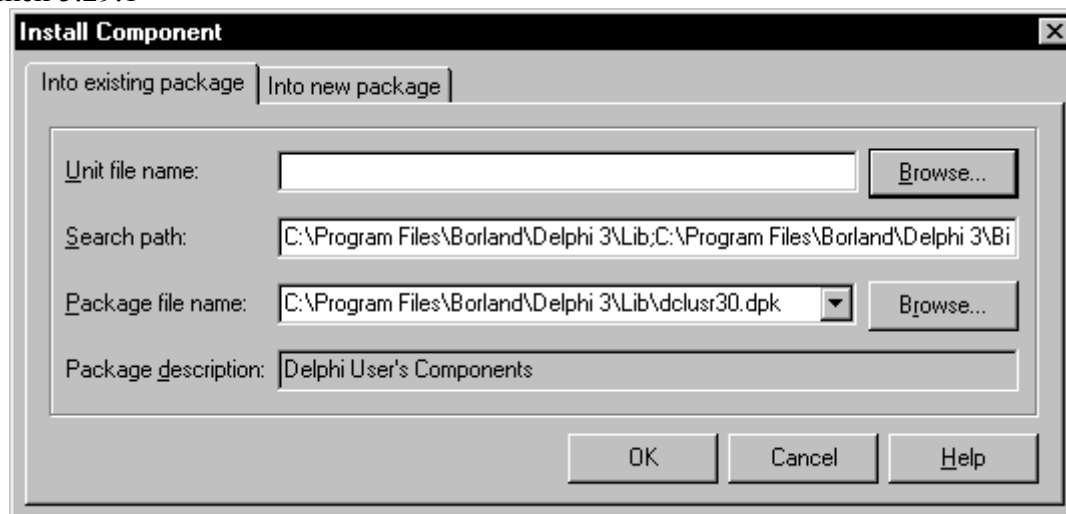
Ćwiczenie 5.29. Instalacja nowych komponentów

Proszę zainstalować nowe komponenty, które znajdują się w katalogu *Delphi\Nowe_Komponenty*. W katalogu tym znajduje się między innymi komponent do odczytywania obrazów JPG, ozdobny klawisz, do wyboru folderu i kilka innych.

Sposób wykonania:

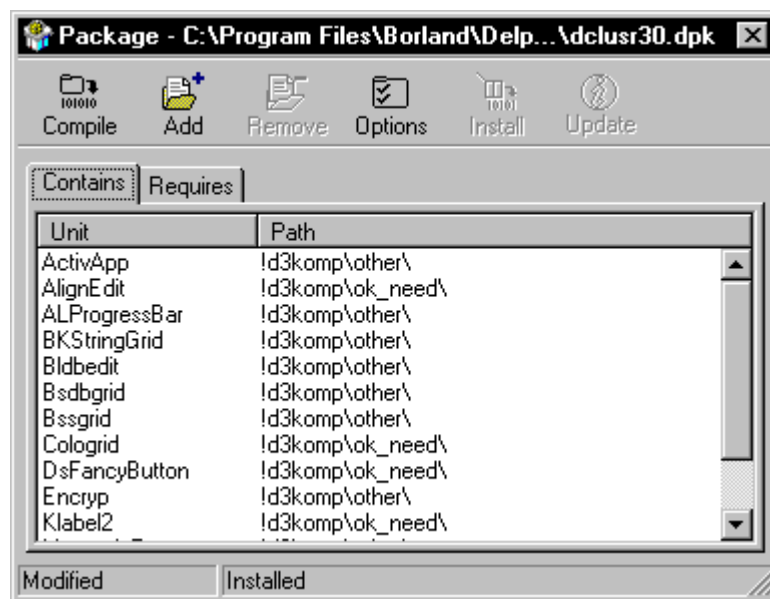
- Wybierz menu „Component” (Komponent) ;
- Wybierz opcję „Install component” (Instaluj komponent);
- Wybierz klawisz „Browse” (Przeglądaj) , który znajduje się obok okienka o nazwie „Unit file name” (Nazwa pliku) – rysunek 5.29.1;

Rysunek 5.29.1



- W ukazanym oknie wybierz nazwę pliku i naciśnij klawisz OK;
- Gdy plik został już wybrany to naciśnij klawisz OK;
- Po dodaniu komponentu zobaczysz okienko (rysunek 5.29.2) z zainstalowanymi komponentami;

Rysunek 5.29.2

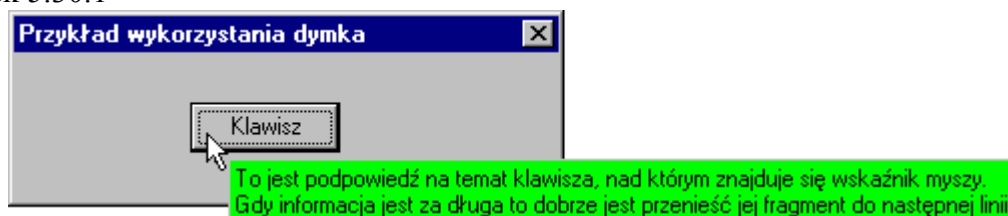


- Zamknij okno przedstawione na rysunku 5.29.2;
- Odpowiedz twierdząco przy pytaniu, który dotyczy zapisania zainstalowanych komponentów;

Ćwiczenie 5.30. Dymki (Podpowiedzi)

Napisz program, który wyświetli opis np. przycisków po najechniu myszką. Rysunek 5.30.1 przedstawia program w działaniu.

Rysunek 5.30.1



Przedstawiony przykład znajduje się w katalogu Delphi\Inne\Dymki.


Opis:

Dymki (podpowiedzi) są komunikatem dla użytkownika mówiącym o przeznaczeniu danej funkcji programu. Komunikaty te są wyświetlane dopiero po najechniu myszką na dany element. Podpowiedzi te znajdują się w małym okienku, którego parametry można zmieniać. Gdy informacja jest za długa, to dobrze jest przenieść jej fragment do następnej linii. Dokonujemy tego przez wstawienie do naszego łańcucha w stosownym miejscu znaku z kodem Enter'a, aby nasza podpowiedź wyświetlona została w następnym wierszu.

Np.

```
Button1.Hint:= 'To jest podpowiedź na temat klawisza, '+CHR(13)+
               'nad którym znajduje się wskaźnik myszy.'+CHR(13)+
               'Gdy informacja jest za długa to dobrze '+CHR(32)+
               'jest przenieść jej fragment do następnej linii';
```

Sposób wykonania:

- Wybierz klawisz **Button**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**) i opisz go zgodnie z rysunkiem 5.30.1;
- Kliknij na formie dwa razy (szybko) 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 odpowiedzią.
```

```
  Application.HintPause:= 100;
```

```
  // Ustawienie czasu, określającego jak długo dymek będzie wyświetlany.
```

```
  Application.HintHidePause:= 5000;
```

```
  Button1.ShowHint:= TRUE; // Włączenie odpowiedzi.
```

```
  // Button1.Hint – Wpisanie odpowiedzi pod konkretny komponent (np. klawisz)
```

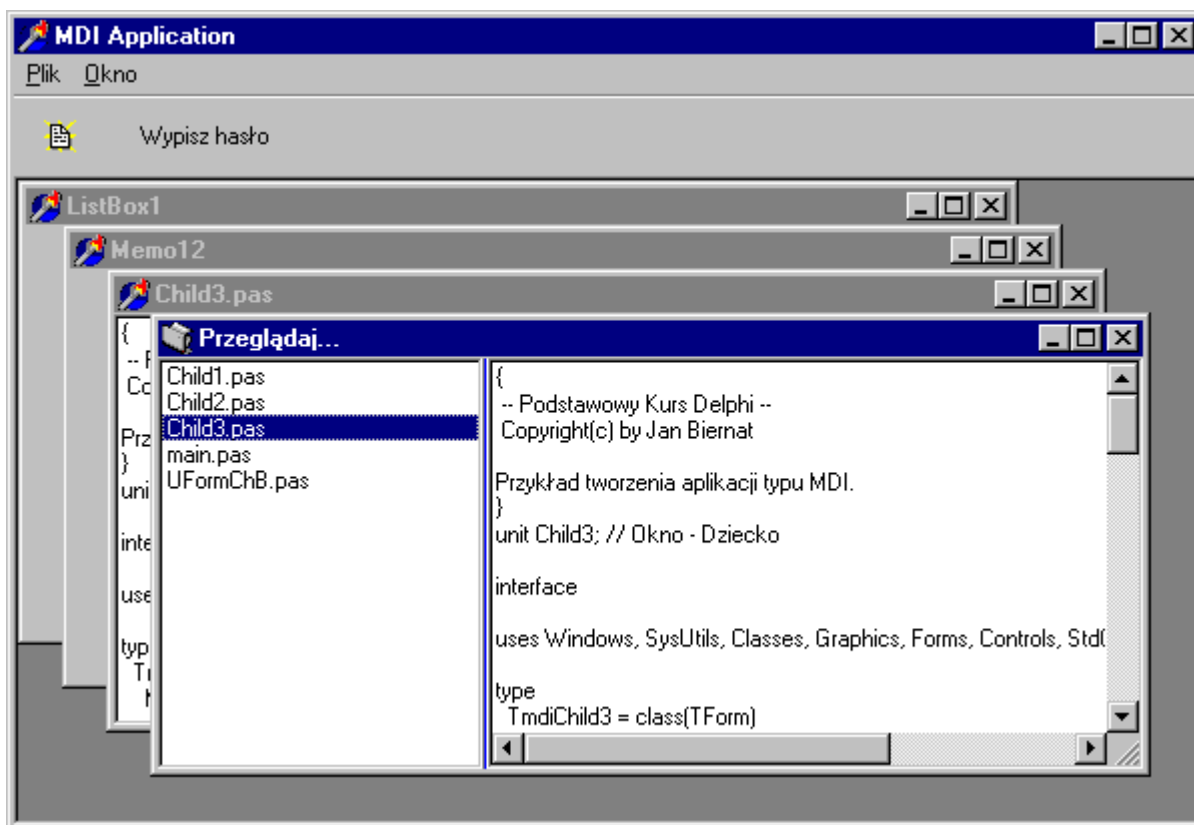
```
  Button1.Hint:= 'To jest odpowiedź na temat klawisza, nad którym znajduje się  
wskaźnik myszy.'+CHR(13)+ 'Gdy informacja jest za długa to dobrze jest przenieść jej  
fragment do następnej linii.';
```

```
end;
```

Ćwiczenie 5.31. MDI (aplikacja wielodokumentowa)

Napisz program, w którym będzie możliwe otwarcie kilku okien naraz. Rysunek 5.31.1 przedstawia program w działaniu.

Rysunek 5.31.1



Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\Mdi.

Opis:

Aplikacje o interfejsie wielodokumentowym mogą jednocześnie otwierać i wyświetlać kilka plików. 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 opcję „New” (Nowy) z menu „File” (Plik);
- Wybierz zakładkę Projects (Projekty);
- Wybierz projekt o nazwie MDI application;
- Zatwierdź klawiszem OK;
- Następnie zostaniesz poproszony o wskazanie katalogu, do którego zostanie zapisany wygenerowany kod;
- Po wybraniu katalogu zatwierdź wybór klawiszem OK.

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.FilePrzeglądajClick(Sender: TObject);  
const  
  // Zadeklarowanie stałej określającej nazwę okienka służącego do przeglądania plików.  
  txtPrzegląd = '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 = txtPrzegląd) then  
      begin  
        // Gdy okno zostanie znalezione, s 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 CreateMDIChildPrzeglądaj(txtPrzegląd);  
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 numeru różnego od zera wybranej opcji menu (np.. menu „OKNO”) we właściwości GroupIndex, co powoduje połączenie się dwóch menu rodzica i dziecka.

Ćwiczenie 5.32. Wczytanie pliku przez podanie jego nazwy jako parametr

Wykonaj program, który wczyta plik lub pliki w momencie uruchomienia przez kliknięcia na nazwie pliku.

Przedstawiony przykład znajduje się w katalogu Delphi\Inne\ParamStr.

Aby przykład zadziałał należy program wywołać za pomocą opcji URUCHOM w menu START pisząc następującą składnię: view.exe egipt.bmp fale.bmp.

Opis:

Możliwość wczytywania pliku do programu przez podanie go jako parametr 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 parametr.

Umożliwia to również skojarzenie plików z konkretną aplikacją, przez co wybór danego pliku (np. w Eksploratorze) spowoduje uruchomienie się konkretnej aplikacji.

Sposób wykonania:

- Kliknij dwa razy (szybko) na formatce i w wygenerowanym zdarzeniu **OnCreate** lub **OnShow** wpisz kod odpowiedzialny za wczytanie pliku przy uruchomieniu programu. W naszym przykładzie kod będzie umieszczony w zdarzeniu **OnShow**:

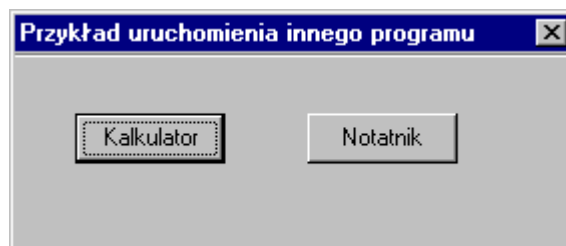
```
procedure TForm1.FormShow(Sender: TObject);
var
  TT: Integer; // Deklaracja zmiennej
begin
  // Wyliczanie ilości parametrów podanych w
  // momencie uruchomienia aplikacji (programu).
  for TT:= 1 to ParamCount do
    if (Trim(ParamStr(TT))<>") then Viewer(Trim(ParamStr(TT)));
  {
    ParamStr(TT) – funkcja zwraca parametr o podanym numerze w wywołanym
    programie.
    Funkcja ParamCount zwraca liczbę parametrów podanych w
    wywołaniu aplikacji. Parametry oddziela się znakiem spacji.

    Przykłady:
    ParamStr(0) - zwraca ścieżkę dostępu i nazwę uruchomionej
    aplikacji (programu).
    ParamStr(1) - zwraca parametr, który był podany w momencie
    wywołania aplikacji (programu).
  }
end;
```

Ćwiczenie 5.33. Uruchomienie innego programu z poziomu aplikacji

Wykonaj program, który będzie uruchamiał program KALKULATOR i NOTATNIK. Rysunek 5.33.1 przedstawia taki program.

Rysunek 5.33.1




Przedstawiony przykład znajduje się w katalogu Delphi\Inne\RunApp.

Opis:

Możliwość uruchomienia innego programu (np. kalkulatora) z naszego programu jest bardzo wygodnym ułatwieniem zwalniającym użytkownika z dodatkowych czynności związanych z uruchomieniem programu (np. kalkulatora w celu dokonania jakichś obliczeń).

Sposób wykonania:

- Wpisz nazwę biblioteki **ShellApi** w deklaracji **Uses**, np. **uses ShellApi** (bez tej deklaracji funkcja `ShellExecute` nie będzie działać);
- Wybierz klawisz **Button**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Kliknij na klawiszu dwa razy (szybko) i w wygenerowanym zdarzeniu wpisz kod:

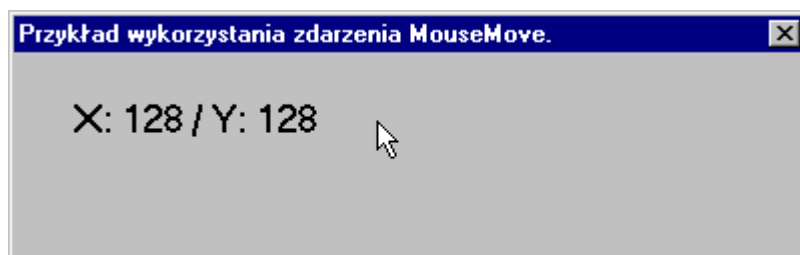
```
procedure TForm1.bUruchomKalkulatorClick(Sender: TObject);  
begin  
    // Uruchomienie programu Kalkulator (calc.exe)  
    ShellExecute(Handle, 'open', 'calc.exe', "", "", sw_Normal);  
end;
```

- Tak samo postępuj przy drugim klawiszu zmieniając nazwę programu.

Ćwiczenie 5.34. Pozycja kursora myszy

Napisz program, który będzie pokazywał pozycję kursora myszy. Rysunek 5.34.1 pokazuje wygląd takiego programu.

Rysunek 5.34.1



Przykład znajduje się w katalogu Delphi\Inne\Pkursora.

Sposób wykonania:

- Wybierz okno **Object Inspector** (Inspektor Obiektów);
- Wybierz zakładkę **Events** (Zdarzenia);
- Wybierz zdarzenie **OnMouseMove** i kliknij na nim dwa razy (szybko);
- W wygenerowanej procedurze wpisz kod:

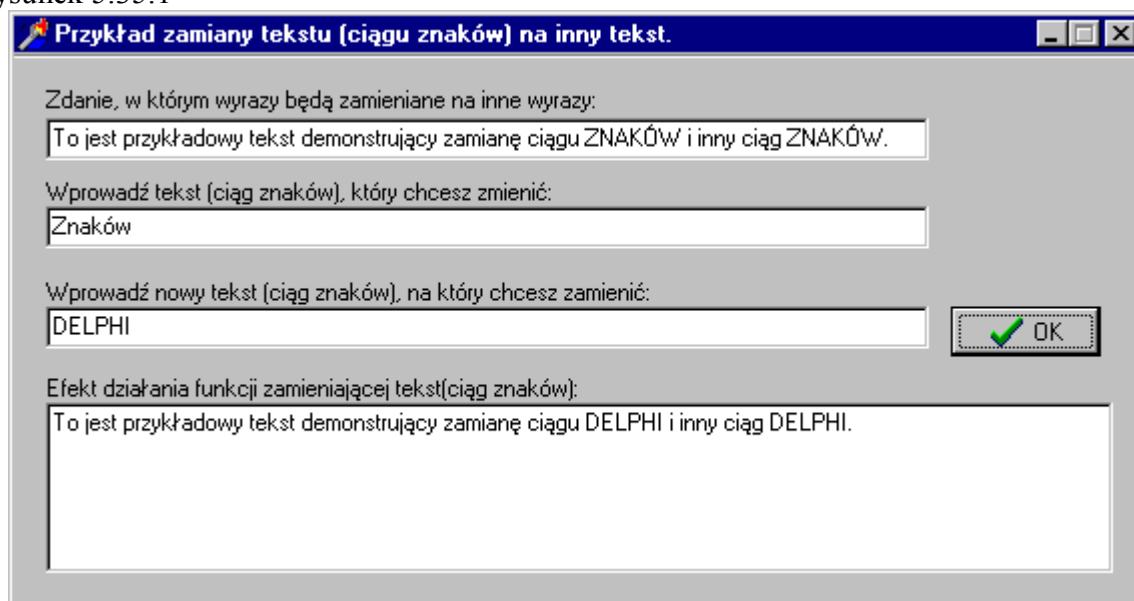
```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
var
  // Zadeklarowanie obiektu reprezentującego punkty na ekranie.
  Position: TPoint;
begin
  // Podanie pozycji kursora myszy w momencie poruszania nim.
  GetCursorPos(Position);

  // 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;
```

Ćwiczenie 5.35. Zamiana znaków w tekście




Napisz program, który będzie zamieniał jeden ciąg znaków na inny ciąg znaków. Rysunek 5.35.1 przedstawia taki program.

Rysunek 5.35.1



Przykład znajduje się w katalogu Delphi\Inne\ zm_znak.

Sposób wykonania:

- Wybierz kilka komponentów **Label**  i kilka komponentów **Edit**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz klawisz **BitBtn**  (karta **Additional**);
- Wybrane komponenty rozmieść na formatce zgodnie z rysunkiem 5.35.1;
- Napisz funkcję zamieniającą jeden ciąg znaków na inny:

```
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  
  ciagu znaków, pętla nie wykona się ani razu.  
}
```

```
// Podawanie pozycji znalezionej tekstu (ciagu znaków)
```

```
numPos:= 0;
```

```
numPos:= Pos(AnsiUpperCase(txtFind), AnsiUpperCase(txtText));
```

```
// Usunięcie znalezionej tekstu (ciagu znaków)
```

```
Delete(txtText, numPos, numLen);
```

```
// Wstawienie nowego tekstu (ciagu znaków) w miejsce starego
```

```
Insert(txtReplace, txtText, numPos);
```

end;

```
jbZnajdzZamien:= txtText;
```

end;

- Zaznacz klawisz z napisem „OK” i kliknij na nim dwa razy (szybko). Po wygenerowaniu procedury wpisz kod:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
```

begin

```
  // Wywołanie funkcji zamieniającej ciągu znaków.
```

```
  Memo1.Lines.Clear; // Wyczyszczenie komponentu Memo1
```

```
  // Dodanie efektu zamiany tekstu (ciagu znaków) do Memo1
```

```
  Memo1.Lines.Add(jbZnajdzZamien(
```

```
    Edit1.Text,
```

```
    Edit2.Text,
```

```
    Edit3.Text
```

```
  ));
```

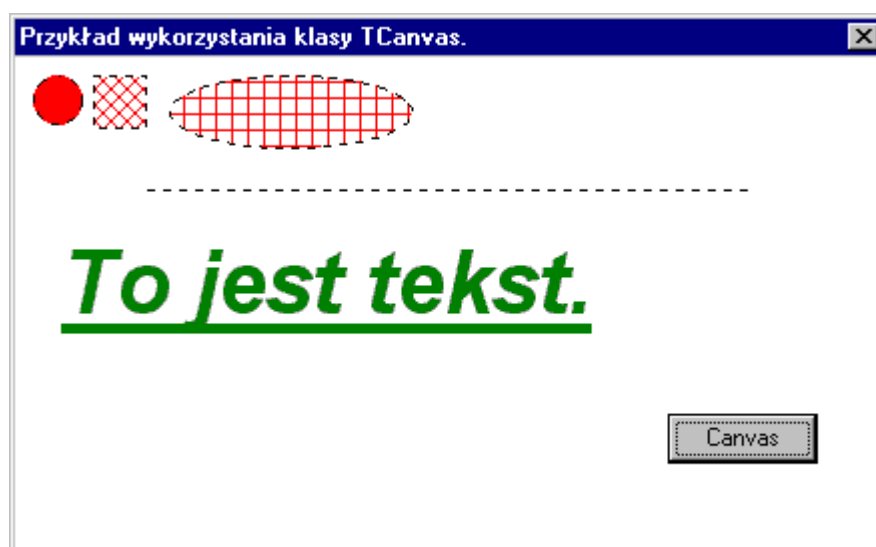
end;

Ćwiczenie 5.36. Grafika

Napisz program, który narysuje figury geometryczne i napisze tekst na formie (płótnie).

Rysunek 5.36.1 przedstawia taki program.

Rysunek 5.36.1




Przykład znajduje się w katalogu Delphi\Inne\Canvas.

Opis:

Każda formatka posiada własną własność Canvas (Płótno), która pochodzi od obiektu TCanvas. Obiekt ten udostępnia funkcje do tworzenia grafiki i wypisywania tekstu.

Sposób wykonania:

- Wybierz komponent **Button**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Zaznacz ten klawisz i kliknij na nim dwa razy szybko;
- W wygenerowanej procedurze wpisz kod:

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

```
begin
```

```
  // Wyczyszczenie obszaru rysowania.
```

```
  with 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;
```

```
  //-- Koło --
```

```
  // Ustalenie koloru jakim będzie narysowana figura.
```

```
Canvas.Pen.Color:= clBlack;

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

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

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

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

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

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

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

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

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

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

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

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

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

// Ustalenie koloru tekstu
```



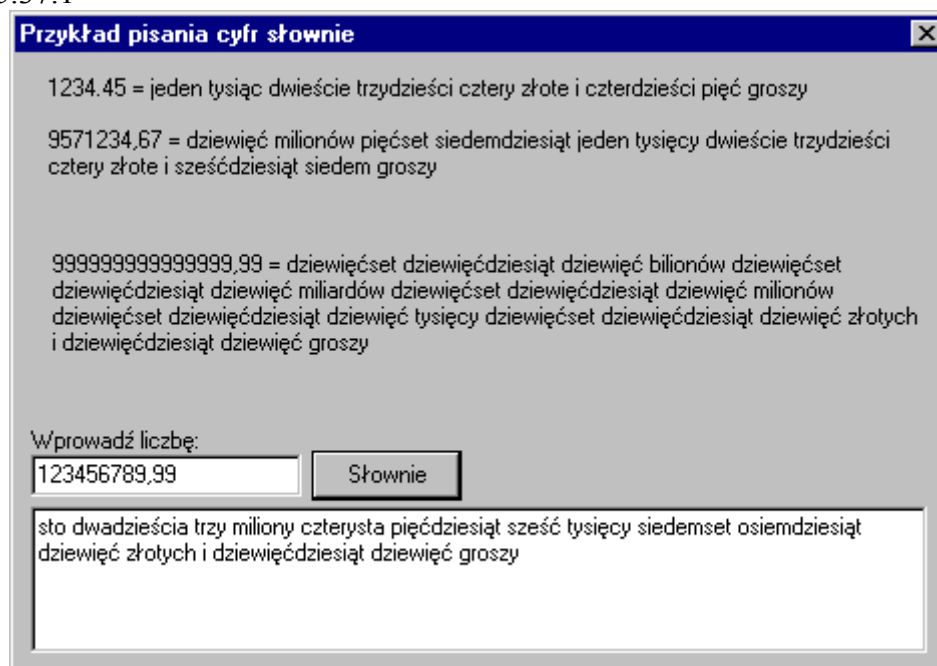
```
Canvas.Font.Color:= clGreen;  
  
// Ustalenie stylu tekstu  
Canvas.Font.Style:= [fsBold, fsItalic, fsUnderline];  
  
// Ustalenie wielkości tekstu  
Canvas.Font.Size:= 33;  
  
// Wypisanie tekstu na ekranie  
Canvas.TextOut(23, 88, 'To jest tekst.');
```

end;

Ćwiczenie 5.37. funkcja słownie




Napisz program, który będzie zamieniał cyfry na słowa z zachowaniem gramatyki. Rysunek 5.37.1 przedstawia taki program.

Rysunek 5.37.1



Przykład znajduje się w katalogu Delphi\Inne\Slownie.

Sposób wykonania:

- Wybierz komponent **Edit**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponent **Memo**  (karta **Standard**);
- Wybierz komponent **Button**  (karta **Standard**) i umieść go na formie z prawej strony komponentu **Edit** i opisz go zgodnie z rysunkiem 5.37.1;
- Kliknij na ten klawisz dwukrotnie i w wygenerowanej procedurze wpisz kod:

```
procedure TfrmMain.bSayClick(Sender: TObject);  
begin  
  // Słownie  
  Memo1.lines.Clear;  
  Memo1.lines.Add(plnSłownie(eNumber.Text));  
  {  
    plnSłownie('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;
```

- Zadeklaruj bibliotekę Słownie w deklaracji **uses**, np. **uses Słownie**;
- W ostatnim kroku napisz bibliotekę do konwersji liczb na słowa. Przykładowa biblioteka może wyglądać tak:

unit Słownie;

interface

uses

SysUtils;

function plnSłownie(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  
przeciwным 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;  
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 txtSłownieGramatyka(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  
  // txtSłownieGramatyka  
  txtSłownieGramatyka:= "";  
  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  
      txtSłownieGramatyka:= MM[mcStrToInt(Copy(txtNumber, 2, 2))-10, 4]+NN[numPos, 3];  
      if (okPenny > 0) then  
        txtSłownieGramatyka:= 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 '+txtSloynieGramatyka(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:= txtSloynieGramatyka(Copy(txtNumber, 1, 3), 5, 0)+
        txtSloynieGramatyka(Copy(txtNumber, 4, 3), 4, 0)+
        txtSloynieGramatyka(Copy(txtNumber, 7, 3), 3, 0)+
        txtSloynieGramatyka(Copy(txtNumber, 10, 3), 2, 0)+
        txtSloynieGramatyka(Copy(txtNumber, 13, 3), 1, 0);
end;
txtSloyniePisz:= txtSay+txtSPenny;
end;

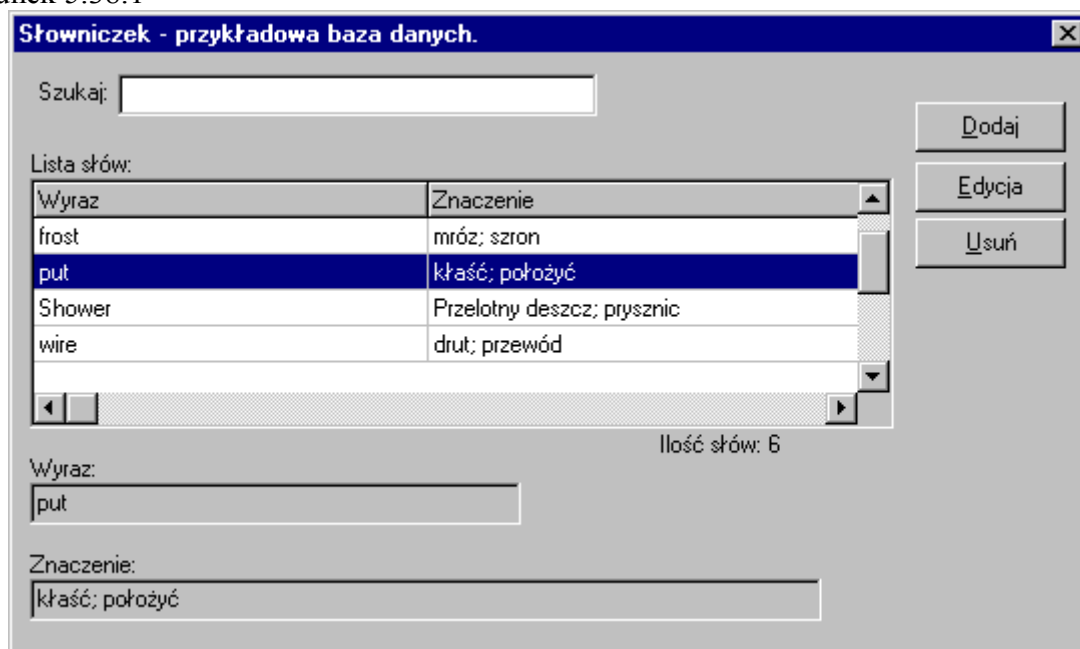
// Wyświetlenie cyfry słownie.
function plnSloynie(txtNumber: String) :String;
var
    numComma: Shortint;
    txtBeforeComma, txtAfterComma: String;
begin
    // plnSloynie
    numComma:= 0;
    numComma:= Pos(',', CommaToDot(txtNumber));
    if (numComma <>0) then
    begin
        txtAfterComma:= "";
        txtAfterComma:= FillZero(Copy(txtNumber, numComma+1, 2));
        txtBeforeComma:= "";
        txtBeforeComma:= Copy(txtNumber, 1, numComma-1);
        plnSloynie:= txtSloyniePisz(txtBeforeComma, txtAfterComma);
    end
    else
        plnSloynie:= txtSloyniePisz(txtNumber, '-1');
    end;

end.
```

Ćwiczenie 5.38. Wprowadzenie do baz danych

Napisz słownik, z możliwością sortowania i wyszukiwania według kolumny WYRAZ. Rysunek 5.38.1 przedstawia wygląd programu.

Rysunek 5.38.1



Przedstawiony przykład znajduje się w katalogu Delphi\Cwicz\Słownik.




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:

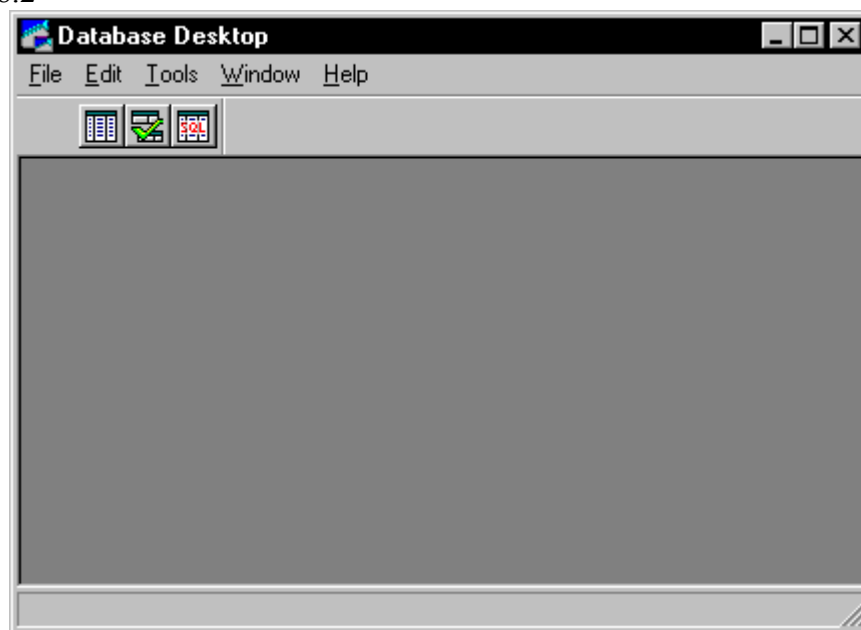
- Wybierz kilka klawiszy **Button**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**) i opisz je zgodnie z rysunkiem 5.38.1;
- Wybierz komponent **Edit**  (karta **Standard**);
- Wybierz kilka komponentów **Label**  (karta **Standard**) i wpisz do nich tekst zgodnie z rysunkiem 5.38.1;

- Wybierz następujące komponenty: - **DBGrid**  (karta **Data Controls**), - **DBEdit**  (karta **Data Controls**);
- Ułóż wyżej wybrane komponenty zgodnie z rysunkiem 5.38.1;
- Wybierz następujące komponenty: - **Query**  (karta **Data Access**), - **DataSource**  (karta **Data Access**);

W celu utworzenia tabeli wykonaj następujące czynności:

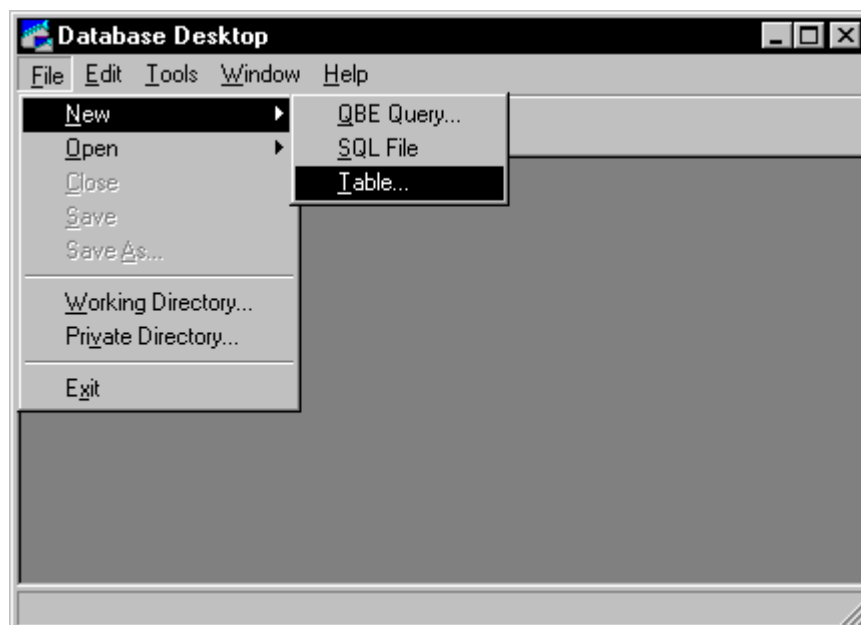
- Wybierz program Database Desktop z menu „Tools” (Narzędzia) – rysunek 5.38.2;

Rysunek 5.38.2



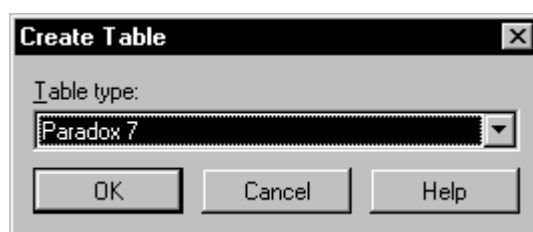
- Wybierz opcję „New” (Nowy) z menu „File” (Plik) w programie Database Desktop;
- Wybierz opcję „Table” (Tabela) z menu „New” (Nowy) – rysunek 5.38.3;

Rysunek 5.38.3



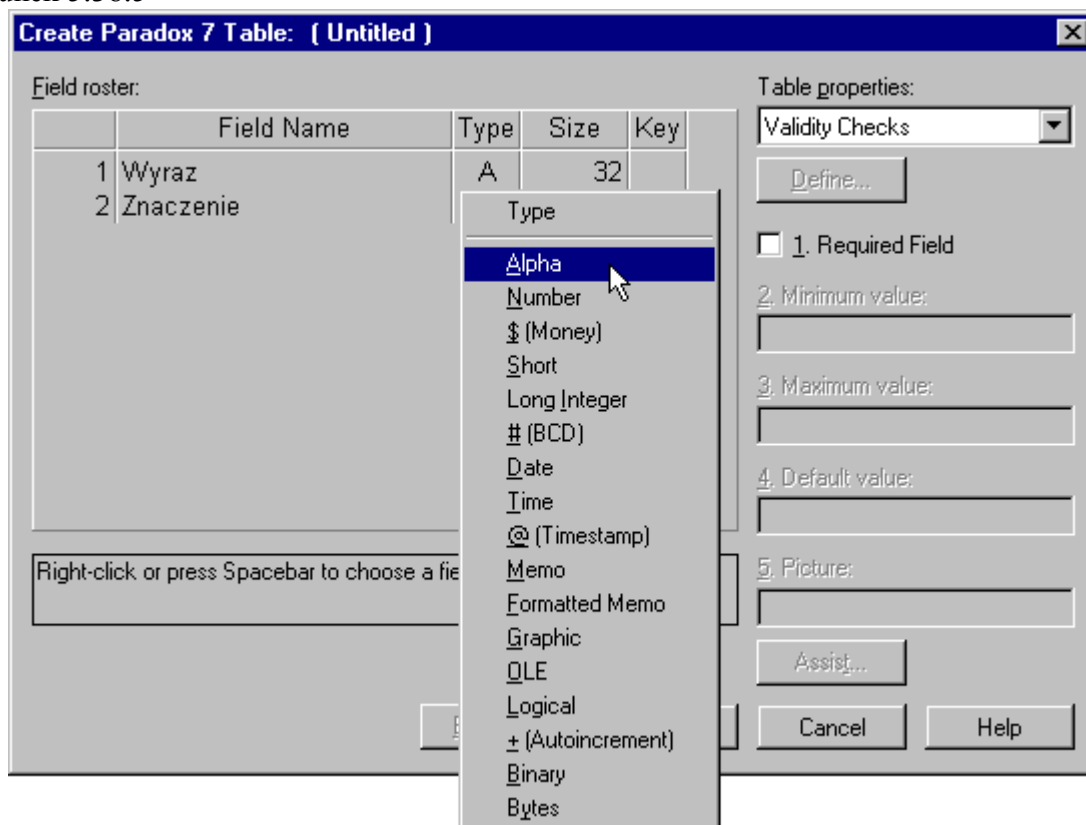
- Wybierz rodzaj bazy, w której chcesz stworzyć tabelę (domyślnie ustawione jest na Paradox 7) – rysunek 5.38.4;

Rysunek 5.38.4



- Zatwierdź wybór klawiszem Enter lub kliknij na klawisz z napisem „OK”, co spowoduje ukazanie się okna (rysunek 5.38.5) w którym będziesz mógł definiować pola;

Rysunek 5.38.5



- 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, które znajdują się na zakładce **Data Access** i **Data Controls** palety komponentów. Użyjemy do tego celu trzech komponentów. Tymi komponentami są **Query**, **DataSource** i **DBGrid**.

Query – jest komponentem reprezentującym zbiór danych, które są wynikiem zapytania SQL (ang. Standard Query Language).

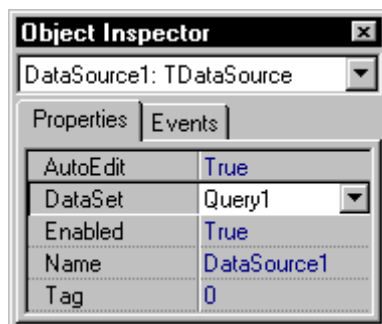
DataSource – jest komponentem realizującym połączenie pomiędzy zbiorami danych (tabelą) a innymi komponentami (np. TDBGrid).

DBGrid – jest komponentem służącym do wyświetlania zawartości zbioru danych (np. tabeli).

Połączenie tych komponentów jest następujące:

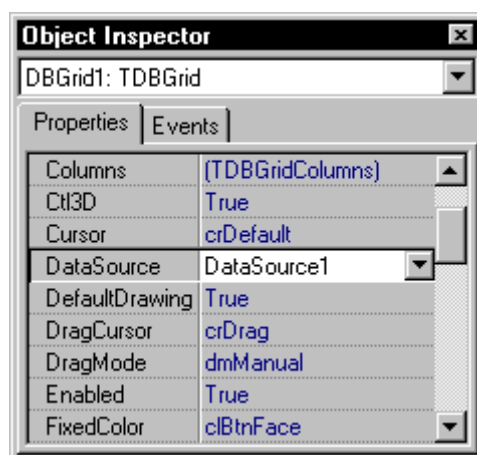
- Połącz **TDataSource** z **TQuery** (np. **Query1**) za pomocą właściwości **DataSet** komponentu **TDataSource** – rysunek 5.38.6;

Rysunek 5.38.6



- Połącz **TDBGrid** z **TDataSource** (np. **DataSource1**) za pomocą właściwości **DataSource** komponentu **TDBGrid** – rysunek 5.38.7.

Rysunek 5.38.7

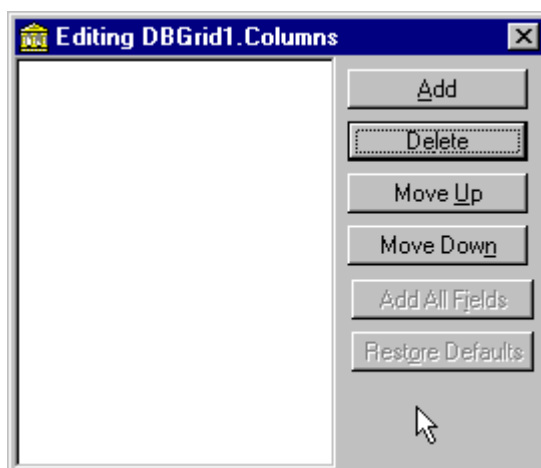


Nazwy kolumn w komponencie DBGrid:

Standardowo nazwy wyświetlane w kolumnach są pobierane z bazy. W celu zmiany opisu kolumn należy wykonać następujące czynności:

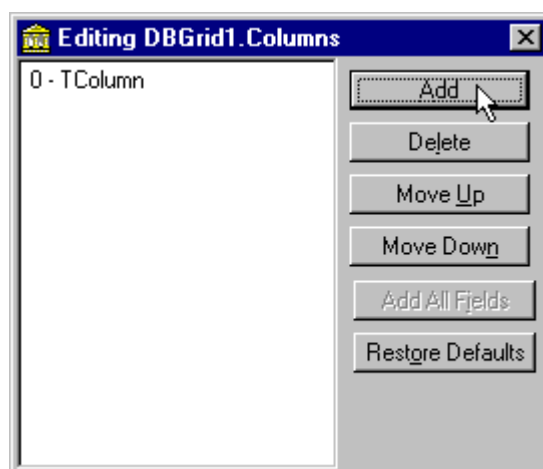
- Kliknij dwukrotnie na komponent **DBGrid** (umieszczony jest na formacie), co spowoduje ukazanie się okna „Editing DBGrid1.Columns” (Edycja kolumn komponentu **DBGrid**) – rysunek 5.38.8;

Rysunek 5.38.8



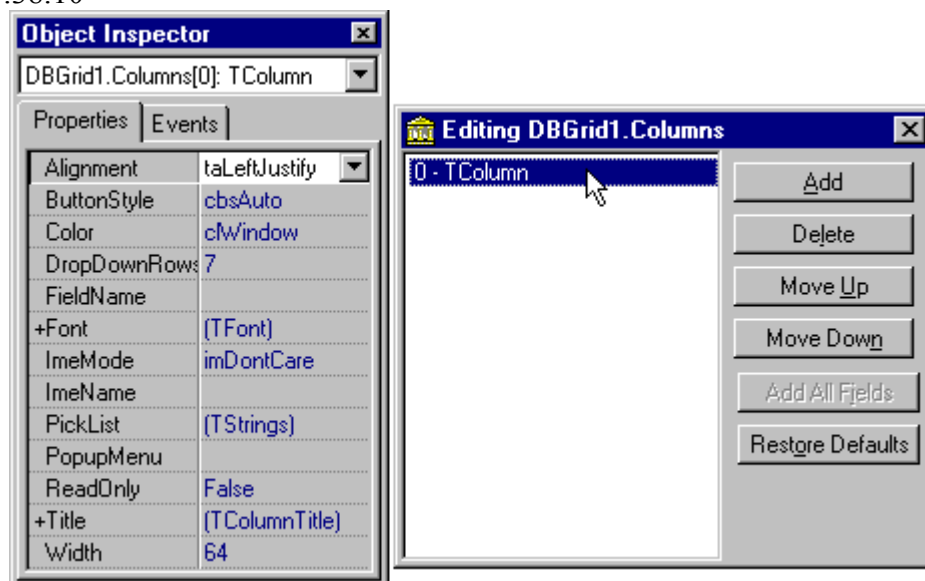
- Kliknij na klawisz z napisem „Add” (Dodaj), co spowoduje ukazanie się nowego elementu na liście – rysunek 5.38.9;

Rysunek 5.38.9



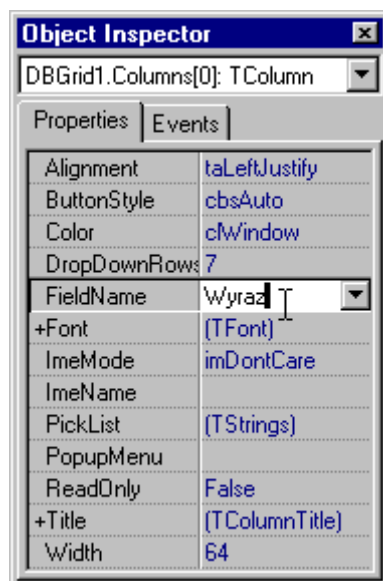
- Zaznacz element na liście, w wyniku czego w oknie Inspektora Obiektów zobaczymy właściwości zaznaczonego elementu – rysunek 5.38.10;

Rysunek 5.38.10



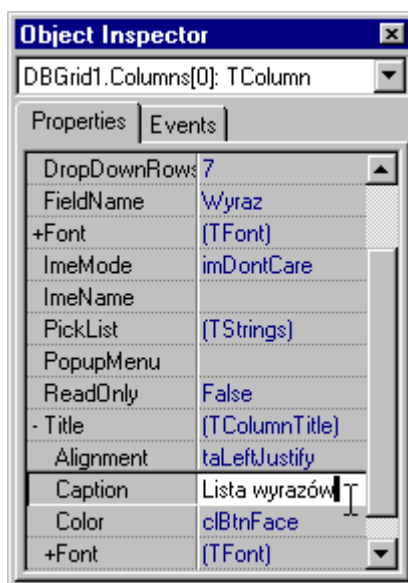
- Wpisz nazwę pola z godną z nazwą kolumny w bazie we właściwości **FieldName** zaznaczonego elementu – rysunek 5.38.11;

Rysunek 5.38.11



- Rozwiń właściwość **Title** przez dwukrotne kliknięcie na tej nazwie;
- Wpisz we właściwości **Caption** tekst „Lista wyrazów” – rysunek 5.38.12;

Rysunek 5.38.12



- Z następnymi elementami listy postępuj tak samo.

Umieszczenie kodu programu:

- Wpisz kod odpowiedzialny za wyświetlenie zawartości bazy (tabeli) w zdarzeniu **OnShow** (wygenerowanie zdarzenia zostało opisane w podrozdziale 1.3).

```

procedure TfrmForm1.FormShow(Sender: TObject);
begin
    // Wyświetlenie zawartości bazy słownika
  
```

```

    Query1.Close; // Zamknięcie bazy danej
    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 "Wraz".

}

Query1.Open; *// Otwarcie bazy danych*

// 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;

- Wybierz klawisz z napisem „Dodaj” i kliknij na nim dwa razy (szybko) i w wygenerowanej procedurze wpisz kod:

procedure TfrmForm1.bDodajClick(Sender: TObject);

begin

// Dodanie nowego słówka (w bazie rekordu).

// Wyświetlenie nagłówka okna edycyjnego.

frmDodajEdycja.Caption:= 'Dodaj';

// Wyczyszczenie zawartości komponentu EDIT w oknie edycyjnym.

frmDodajEdycja.Edit1.Text:= '';

```
// Wyczyszczenie zawartości komponentu EDIT w oknie edycyjnym.
frmDodajEdycja.Edit2.Text:= "";

// Wywołanie okienka edycyjnego do wprowadzenia nowego słowa.
frmDodajEdycja.ShowModal;

{
  Sprawdzenie czy zostało wpisane nowe słówko,
  jeżeli tak, do dodaj to słówko do tabeli.
}
if (Trim(frmDodajEdycja.Edit1.Text) <> "") then
begin
  // Dodanie nowego słowa (nowego rekordu).
  Query1.Close;
  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(frmDodajEdycja.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(frmDodajEdycja.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 słowa.
  }
}
```

```
Query1.ExecSQL; // Wykonanie zapytania SQL.

FormShow(Sender);
{
  FormShow(Sender); - Wywołanie funkcji odczytującej tabelę w celu
                        odświeżenia wyświetlanej informacji.
}
end;
end;
```

- Wybierz klawisz z napisem „Edycja” i kliknij na nim dwa razy (szybko) i w wygenerowanej procedurze wpisz kod:

```
procedure TfrmForm1.bEdycjaClick(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.
  frmDodajEdycja.Caption:= 'Edycja';

  // Odczytanie wiersza z komponentu DBGrid (wyraz). Kolumna 0.
  frmDodajEdycja.Edit1.Text:= DBGrid1.Fields[0].AsString;

  // Odczytanie wiersza z komponentu DBGrid (znaczenie). Kolumna 1.
  frmDodajEdycja.Edit2.Text:= DBGrid1.Fields[1].AsString;

  // Wywołanie okienka edycyjnego do wprowadzenia nowego słowa.
  frmDodajEdycja.ShowModal;

  // Sprawdzenie czy zostało wpisane nowe słowo.
  if (Trim(frmDodajEdycja.Edit1.Text) <> "") then
  begin
    Query1.Close;
    Query1.SQL.Clear;
    Query1.SQL.Add('UPDATE slownik.db SET Wyraz = :p0, Znaczenie = :p1 '+'
                    'WHERE ID = :p2');
    Query1.Params[0].AsString:= Trim(frmDodajEdycja.Edit1.Text); // Wyraz
    Query1.Params[1].AsString:= Trim(frmDodajEdycja.Edit2.Text); // Znaczenie
    Query1.Params[2].AsString:= Trim(txtID); // Pole identyfikacyjne
    Query1.ExecSQL; // Wykonanie zapytania SQL.

    FormShow(Sender);
  {
```



```
    FormShow(Sender); - Wywołanie funkcji odczytującej tabelę w celu  
                          odświeżenia wyświetlanej informacji.  
  }  
end;  
end;
```

- Wybierz klawisz z napisem „Usuń” i kliknij na nim dwa razy (szybko) i w wygenerowanej procedurze wpisz kod:

```
procedure TfrmForm1.bUsunClick(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;  
  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 odczytującej tabelę w celu  
                          odświeżenia wyświetlanej informacji.  
  }  
end;
```

- Wybierz komponent **Edit** (znajdujący się na formatce) i kliknij na nim dwa razy (szybko) i w wygenerowanej procedurze wpisz kod:

```
procedure TfrmForm1.Edit1Change(Sender: TObject);  
begin  
  // Wyszukanie wyrazu (słówka).  
  Query1.Close;  
  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.  
  }  
end;
```

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 Wyrz).

% - 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))+'%'; // Pole WYRAZ

Query1.Open; // Otwarcie bazy danych do odczytu.

end;

Ćwiczenie 5.39. Wyświetlenie tabeli w komponencie StringGrid

Napisz program, który wyświetli zawartość bazy (Tabeli) w komponencie **StringGrid** i będzie umożliwiał wyszukanie stolicy państwa. Rysunek 5.39.1 przedstawia taki program.

Rysunek 5.39.1









Przykład znajduje się w katalogu Delphi\Cwicz\StringGrid_SQL.

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**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) 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 ma nazwę „Szukaj”;
- Zaznacz komponent **Label** i we właściwości **Caption** wpisz „Wpisz stolicę”;
- Wybierz komponent **Edit**  (karta **Standard**) i kliknij na komponencie **GroupBox1**, który ma nazwę „Szukaj”;
- Wybierz komponent **GroupBox**  (karta **Standard**) i we właściwości **Caption** wpisz wyraz „Lista”;
- Wybierz komponent **StringGrid**  (karta **Additional**) i kliknij na komponencie **GroupBox2**, który ma nazwę „Lista”;
- Kliknij dwukrotnie na formacie, co spowoduje wygenerowanie zdarzenia **OnCreate**. 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;

```

- W oknie **Object Inspector** (Inspektor obiektów) wybierz zakładkę **Events** (Zdarzenia) i kliknij dwukrotnie na zdarzeniu **OnShow** 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 bazy danej
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 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".

  ORDER BY - klauzula ta służy do sortowania według wybranej kolumny.
            W naszym przykładzie jest to kolumna "Name".
}

Query1.Open; // Otwarcie bazy danej

// 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;
```

➤ Zaznacz komponent **Edit** i kliknij na nim dwukrotnie (szybko) oraz wprowadź kod w wygenerowanej procedurze **OnChange**:

```
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 bazy danych
  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 danej

// 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.FieldName('Name').AsString);

  {
    Odczytanie Stolicy z kolumny CAPITAL i przypisanie
    jej do drugiej kolumny komponentu StringGrid.
  }
  StringGrid1.Cells[1, 1+TT]:= Trim(Query1.FieldName('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;
```

Ćwiczenie 5.40. Sortowanie w komponencie StringGrid

Napisz program, który wyświetli dane takie jak: Nazwisko, Imię, Adres i Miasto w postaci tabeli. Program powinien posiadać możliwość posortowania tabeli według wybranej kolumny. Do wykonania ćwiczenia należy wykorzystać komponent **StringGrid**. Rysunek 5.40.1 przedstawia przykładowy wygląd programu.

Rysunek 5.40.1








Przykład znajduje się w katalogu Delphi\Cwicz\StrGrid_Sort.


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**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Additional**);
- Wybierz dwa klawisze **Button**  (karta **Standard**) i nazwij je „Wyjście” i „Wypełnij”;
- Wybierz komponent **GroupBox**  (karta **Standard**) i we właściwości **Caption** wpisz wyraz „Sortowanie”;
- Wybierz komponent **Label** **A**  (karta **Standard**) i kliknij na komponent **GroupBox** oraz we właściwości **Caption** wpisz wyraz „Kolumna”;
- Wybierz komponent **ComboBox**  (karta **Standard**) i kliknij na komponent **GroupBox**;

- Wybierz klawisz **Button**  (karta **Standard**) i kliknij na komponent **GroupBox** oraz we właściwości **Caption** komponentu **Button** wpisz „Sortowanie”;
- Kliknij dwukrotnie (szybko) na formatce i w wygenerowanym zdarzeniu **OnCreate** wpisz kod opisujący kolumny:

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

```
begin
```

```
  // Wpisanie nazw kolumn
```

```
  StringGrid1.Cells[0, 0] := 'Nazwisko';
```

```
  StringGrid1.Cells[1, 0] := 'Imię';
```

```
  StringGrid1.Cells[2, 0] := 'Adres';
```

```
  StringGrid1.Cells[3, 0] := 'Miasto';
```

```
  // Wyczyszczenie edytora komponentu ComboBox
```

```
  ComboBox1.Text := '';
```

```
end;
```

- Wybierz klawisz z napisem „Wypełnij” i kliknij na nim dwukrotnie oraz wpisz kod do wygenerowanej procedury:

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

```
begin
```

```
  // Wypełnienie tabeli
```

```
  StringGrid1.Cells[0, 1] := 'Dziedzic';
```

```
  StringGrid1.Cells[1, 1] := 'Damian';
```

```
  StringGrid1.Cells[2, 1] := 'Akademii Umiejętności';
```

```
  StringGrid1.Cells[3, 1] := 'Bielsko-Biała';
```

```
  StringGrid1.Cells[0, 2] := 'Roland';
```

```
  StringGrid1.Cells[1, 2] := 'Tadeusz';
```

```
  StringGrid1.Cells[2, 2] := 'Błotna';
```

```
  StringGrid1.Cells[3, 2] := 'Bielsko-Biała';
```

```
  StringGrid1.Cells[0, 3] := 'Wojna';
```

```
  StringGrid1.Cells[1, 3] := 'Marcin';
```

```
  StringGrid1.Cells[2, 3] := 'Siewna';
```

```
  StringGrid1.Cells[3, 3] := 'Bielsko-Biała';
```

```
  StringGrid1.Cells[0, 4] := 'Wąsowski';
```

```
  StringGrid1.Cells[1, 4] := 'Mirek';
```

```
  StringGrid1.Cells[2, 4] := 'Modlińska';
```

```
  StringGrid1.Cells[3, 4] := 'Wrocław';
```

```
  StringGrid1.Cells[0, 5] := 'Gumiś';
```

```
  StringGrid1.Cells[1, 5] := 'Arek';
```


```
  StringGrid1.Cells[2, 5] := 'Malownicza';
```

```
  StringGrid1.Cells[3, 5] := 'Wrocław';
```

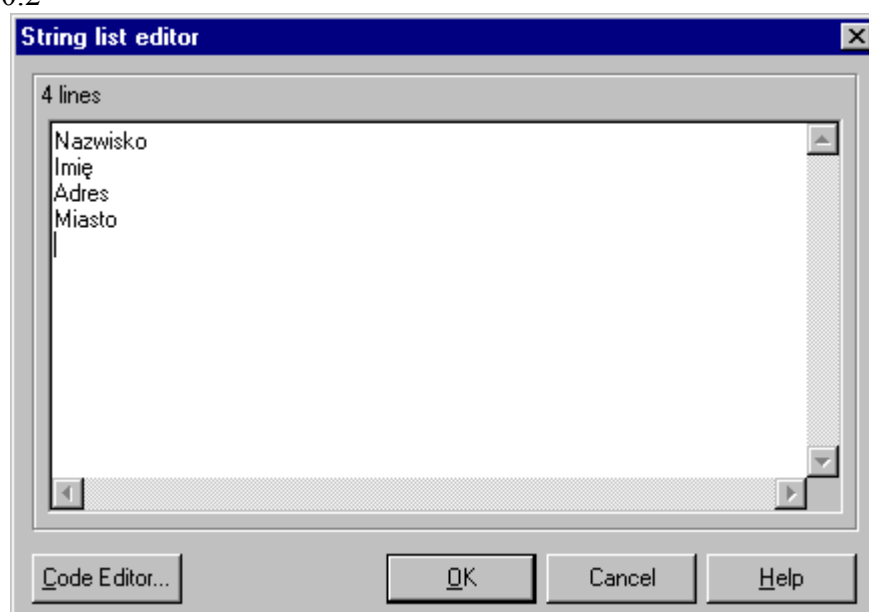
```
end;
```


- Wybierz klawisz z napisem „Wyjście” i kliknij na nim dwukrotnie oraz wpisz kod w wygenerowanej procedurze:

```
procedure TForm1.Button3Click(Sender: TObject);  
begin  
    Close; // Wyjście  
end;
```

- Zaznacz komponent **ComboBox** i przejdź do okna **Object Inspector** (Inspektor obiektów) przez zaznaczenie myszką tego okna lub przez naciśnięcie klawisz F11.
- Wybierz zakładkę **Properties** (Właściwości) i kliknij na właściwość **Items** komponentu **ComboBox**;
- Kliknij na klawisz , co spowoduje ukazanie się okna „String List editor” (Edytor list ciągu znaków), w którym wpisz nazwy kolumn – rysunek 5.40.2;

Rysunek 5.40.2



- Zatwierdź klawiszem „OK” wpisane nazwy;
- Napisz funkcję sortującą:

```
function TForm1.StringGridSort(numCol, numRowsMax: Integer; okTitle: Shortint):  
Integer;  
var  
    AA, BB, TT: Integer;  
    txtSwap: String;  
begin  
    // StringGridSort  
    if (okTitle < 0) then okTitle:= 0;  
    if (okTitle > 1) then okTitle:= 1;  
  
    //--- Sort ---  
    for AA:= numRowsMax downto okTitle do  
    begin
```

```

txtSwap:= "";
txtSwap:= StringGrid1.Cells[numCol, AA];

for BB:= numRowsMax downto okTitle do
begin
  if (CompareText(txtSwap, StringGrid1.Cells[numCol, BB]) >0) then
    begin

      // Swap
      for TT:= 0 to StringGrid1.ColCount-1 do
        begin
          txtSwap:= "";
          txtSwap:= StringGrid1.Cells[TT, AA];

          StringGrid1.Cells[TT, AA]:= StringGrid1.Cells[TT, BB];
          StringGrid1.Cells[TT, BB]:= txtSwap;
        end;

      Break;
    end;
  end;
end;
StringGridSort:= numRowsMax;
end;

```

- Zadeklaruj funkcje w typie obiektowym:

```

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    .....
    function StringGridSort(numCol, numRowsMax: Integer; okTitle: Shortint): Integer;
    procedure FormCreate(Sender: TObject);
    .....
  private
    { Private declarations }
  public
    { Public declarations }
  end;

```

- Wybierz klawisz z napisem „Sortowanie” i kliknij na nim dwukrotnie oraz wpisz kod w wygenerowanym zdarzeniu:

```

procedure TForm1.bSortClick(Sender: TObject);
var
  TT, numCol: Integer;
begin
  // Wywołanie funkcji do sortowania
  numCol:= 0;

```

```

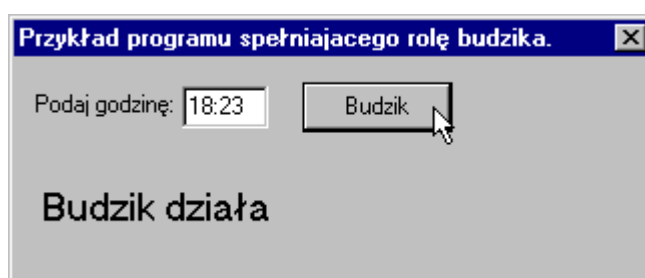
if (ComboBox1.Text = 'Nazwisko') then numCol:= 0;
if (ComboBox1.Text = 'Imię') then numCol:= 1;
if (ComboBox1.Text = 'Adres') then numCol:= 2;
if (ComboBox1.Text = 'Miasto') then numCol:= 3;
for TT:= 0 to StringGrid1.RowCount-1 do
  StringGridSort(numCol, StringGrid1.RowCount-1, 1);
end;

```

Ćwiczenie 5.41. Alarm

Napisz program, który będzie wyświetlał komunikat "Budzik działa", gdy czas wprowadzony będzie zgodny z czasem systemowym. Rysunek 5.41.1 przedstawia przykładowy program.

Rysunek 5.41.1



Przykład znajduje się w katalogu Delphi\Inne\Alarm.

Sposób wykonania:

- Wybierz komponent **Label A** (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz okno **Object Inspector** (Inspektor obiektów) naciskając klawisz funkcyjny F11 i we właściwości **Caption** komponentu **Label** wpisz „Podaj godzinę.”;
- Wybierz klawisz **Button OK** (karta **Standard**) i nazwij go według rysunku 5.41.1;
- Wybierz komponent **Label A** (karta **Standard**) i ustaw ten komponent według rysunku 5.41.1;
- Kliknij dwukrotnie (szybko) na formatce i w wygenerowanym zdarzeniu **OnCreate** wpisz kod:

```

procedure TForm1.FormCreate(Sender: TObject);
begin
  // Wyczyszczenie zawartości komponentu Edit
  MaskEdit1.Text:= "";
end;

```

- Napisz funkcje potrzebne do sprawdzania czasu systemowego (powyżej zdarzenia, np. **OnCreate** = procedurze **FormCreate**, a poniżej słowa **implementation**):

```

function FillZero(txtZero: String): String;
{
  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

}

begin

FillZero:= txtZero;

if (Length(txtZero)=1) **then** FillZero:= '0'+txtZero;

end;

function jbCzas(txtTime: **String**): Shortint;

{

Funkcja zwraca wartość 1, gdy podana

godzina zgadza się z godziną zegara

systemowego. W innym przypadku

zwraca wartość -1.

}

var

// Zadeklarowanie zmiennych tekstowych

txtAHour, txtAMin, txtHour, txtMin: **String**;

begin

//jbCzas

jbCzas:= -1;

// Odczytanie liczb wskazujących godzinę

txtAHour:= "";

txtAHour:= Copy(TimeToStr(Time), 1, 2);

{

Copy(Tekst, Od_Jakieg_Znaku, Do_Jakiego_Znaku)

Wycina fragment ciągu znaków

Time - Zwraca bieżący czas

TimeToStr() - Konwertuje czas na tekst

}

// Odczytanie liczb wskazujących minutę

txtAMin:= "";

txtAMin:= Copy(TimeToStr(Time), 4, 2);

{

Sprawdzenie czy zmienna 'txtTime' jest różna od pustego.

Jeżeli tak to wykonaj instrukcje po słowie THEN, w

przeciwnym przypadku nie wykonuj nic.

}

if (Trim(txtTime)<>") **then**

begin

// Odczytanie liczb wskazujących godzinę

txtHour:= "";

txtHour:= FillZero(Copy(txtTime, 1, 2));

// Odczytanie liczb wskazujących minutę

```
txtMin:= " ";  
txtMin:= FillZero(Copy(txtTime, 4, 2));  
  
{  
  Jeżeli podany czas jest zgodny z czasem  
  systemowym to warunek jest spełniony.  
  W innym przypadku warunek nie jest spełniony.  
}  
if (txtHour+'.'+txtMin = txtAHour+'.'+txtAMin) or  
  (txtHour+'.'+txtMin = '--.'+txtAMin) then jbCzas:= 1;  
end;  
end;
```

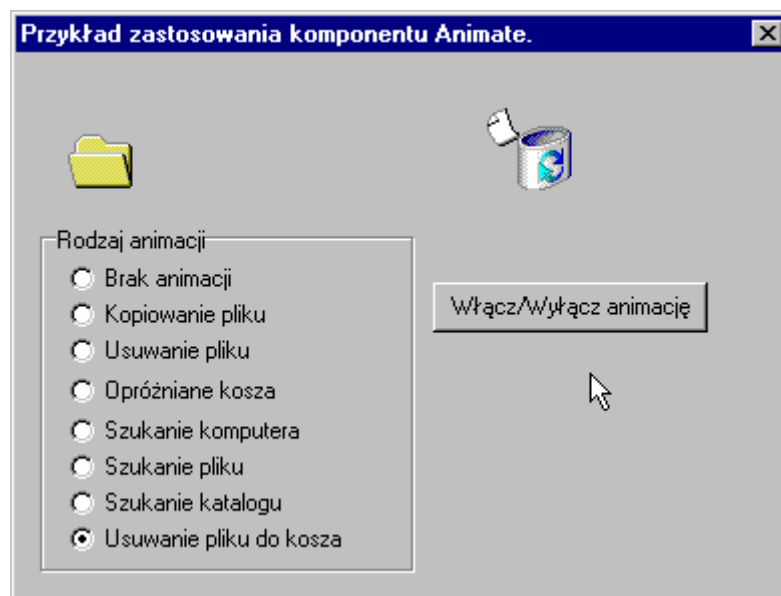
- Wybierz klawisz z napisem „Budzik” i kliknij na nim dwukrotnie (szybko) oraz wpisz kod w wygenerowanej procedurze:

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
  {  
    Sprawdzenie czy czas podany zgadza  
    się z czasem systemowym, jeżeli tak  
    to wyświetlony będzie odpowiedni komunikat..  
  }  
  if (jbCzas(MaskEdit1.Text) = 1) then  
    Label2.Caption:= 'Budzik działa' // Komunikat  
  else  
    Label2.Caption:= " "; // Wyczyszczenie komunikatu  
end;
```

Ćwiczenie 5.42. Standardowe animacje w systemie Windows


Napisz program, który zaprezentuje standardowe animacje występujące w systemie Windows. Rysunek 5.42.1 przedstawia taki program.

Rysunek 5.42.1







Przykład znajduje się w katalogu Delphi\Cwicz\Animate.

Opis komponentu:

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

Sposób wykonania:

- Wybierz komponent **Animate**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Win32**);
- Wybierz komponent **Button**  (karta **Standard**) i opisz go zgodnie z rysunkiem 5.42.1;
- Wybierz komponent **GroupBox**  (karta **Standard**) i we właściwości **Caption** wpisz wyraz „Rodzaj animacji”;
- Wybierz komponent **RadioButton**  i kliknij na komponent **GroupBox** oznaczony jako „Rodzaj animacji”;
- Ze wszystkimi komponentami **RadioButton** postępuj tak samo, przez rozmieszczenie i opisanie tych komponentów zgodnie z rysunkiem 5.42.1;
- Zaznacz pierwszy komponent **RadioButton** oznaczonym jako „Brak animacji” i ustaw jego właściwość **Checked** na TRUE (aktywny) będąc w oknie **Object Inspector** (Inspektor obiektów) oraz kliknij dwukrotnie na nim i w wygenerowanej procedurze wpisz kod:

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

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

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

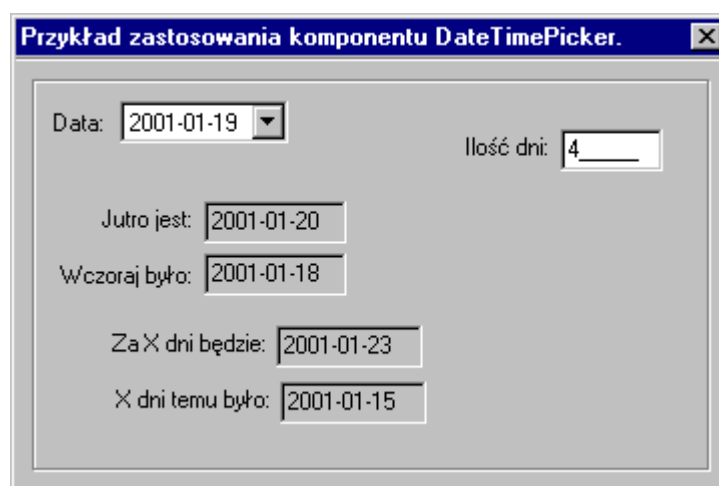
- Z następnymi komponentami **RadioButton** postępuj tak samo jak przy drugim komponencie **RadioButton** z tą różnicą, że właściwości **CommonAVI** będziesz przypisywał inne animacje (np. aviEmptyRecycle, aviFindFolder);
- Kliknij dwukrotnie na klawisz z napisem „Włącz/Wyłącz animację” i w wygenerowanej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

Ćwiczenie 5.43. 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. Przykładowy program przedstawiony jest na rysunku 5.43.1.

Rysunek 5.43.1






Przykład znajduje się w katalogu Delphi\Cwicz\Data.

Opis komponentu:

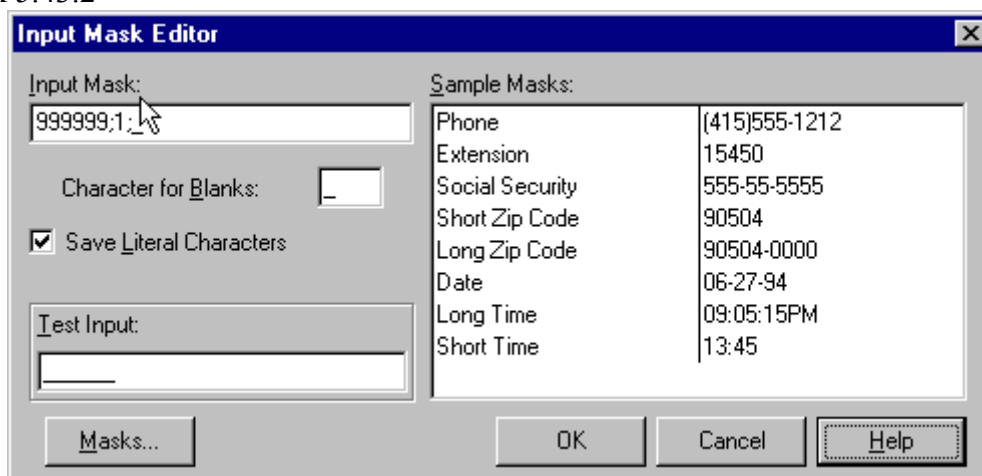


Komponent **DateTimePicker** służy do wybierania daty za pomocą rozwijanego kalendarza oraz czasu.

Sposób wykonania:

- Wybierz komponent **DateTimePicker**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Win32**);
- Wybierz kilka komponentów **Label**  (karta **Standard**) oraz kilka komponentów **Edit**  (karta **Standard**) i ułóż je na formacie zgodnie z rysunkiem 5.43.1;
- Komponenty **Label** opisz zgodnie z rysunkiem 5.43.1;
- 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 formacie za pomocą klawisza TAB);
- Wybierz komponent **MaskEdit**  (karta **Additional**);
- Zaznacz komponent **MaskEdit** i naciśnij klawisz funkcyjny F11. Będąc na zakładce **Properties** (Właściwości) okna **Object Inspector** (Inspektor Obiektów) wybierz właściwość **EditMask** (Edycja maski) przez kliknięcie na klawisz . W ukazanym oknie w sekcji **Input Mask** (Wprowadź maskę) wpisz „999999;1;_” – rysunek 5.43.2;

Rysunek 5.43.2



- Napisz funkcje odpowiedzialne za dodawanie i odejmowanie dni od wprowadzonej daty (powyżej zdarzenia, np. **OnCreate** = procedurze **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;

    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 przesun 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:

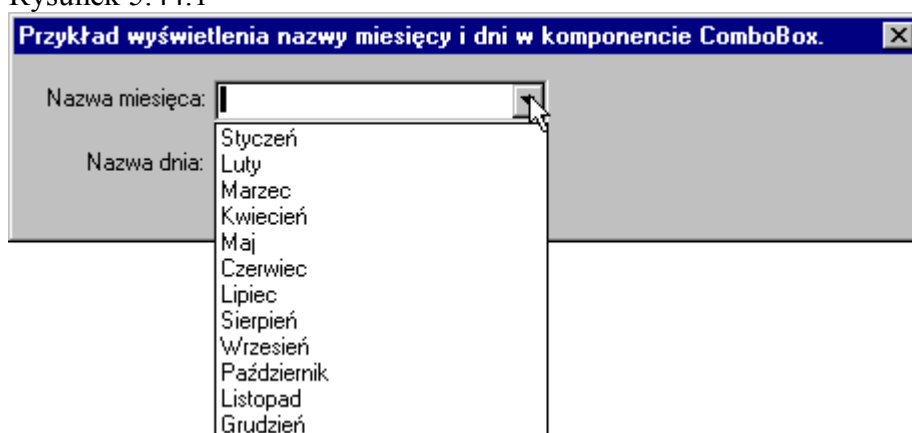
```
procedure TForm1.DateTimePicker1Change(Sender: TObject);
begin
  // Wywołanie poszczególnych funkcji w momencie zmiany
  Edit1.Text:= txtJutroJest(DateToStr(DateTimePicker1.Date));
```

```
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;
```

Ćwiczenie 5.44. Nazwy dni i miesięcy

Napisz program, który wczyta do listy nazwy dni i miesięcy. Nazwy te muszą zaczynać się od dużej litery. Należy użyć komponent **ComboBox**. Rysunek 5.44.1 przedstawia przykładowy program.

Rysunek 5.44.1



Przykład znajduje się w katalogu Delphi\Inne\Nazwy.

Opis funkcji do wyświetlania nazw dni i miesięcy:

Funkcje służące do wyświetlania nazw dni i miesięcy są globalnymi zmiennymi zależnymi od wersji językowej systemu Windows. Tymi zmiennymi są:

- ❖ **LongDayNames[Index]** - Tablica pełnych nazw dni;
- ❖ **LongMonthNames[Index]** - Tablica pełnych nazw miesięcy.

Parametr **Index** decyduje o wyświetlonej nazwie dnia lub miesiąca i przyjmuje następujące wartości:



- ❖ Dla dni od 1 do 7;
- ❖ Dla miesięcy od 1 do 12.

Opis komponentu:



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

Sposób wykonania:

- Wybierz dwa komponenty **ComboBox**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz dwa komponenty **Label**  (karta **Standard**);
- Opisz i ustaw te komponenty zgodnie z rysunkiem 5.44.1;
- Napisz funkcję, która będzie ustawiać napisy tak, aby pierwsza litera była duża, a reszta liter była mała (powyżej zdarzenia, np. **OnCreate** = procedurze **FormCreate**, a poniżej słowa **implementation**):

```
function WyrzDLP(txtText: String): String;  
begin
```

```
{  
  Funkcja konwertuje pierwsza literę ciągu  
  znaków na dużą, natomiast następne znaki  
  zmienia na małe litery.
```

```
  Przykład:
```

- ```
 1) atari -> Atari
 2) delphi -> Delphi
```

```
}
```

```
 WyrzDLP:= AnsiUpperCase(Copy(Trim(txtText), 1, 1))+
 AnsiLowerCase(Copy(Trim(txtText), 2, Length(Trim(txtText))-1));
```

```
{
 AnsiUpperCase() - Zamienia ciąg znaków na ciąg
 znaków pisany dużymi literami
```

```
 AnsiLowerCase() - Zamienia ciąg znaków na ciąg
 znaków pisany małymi literami
```

```
 Copy(Tekst, Od_Jakieg_Znaku, Do_Jakiego_Znaku)
 Wycina fragment ciągu znaków
```

```
 Length() - Oblicza liczbę znaków w tekście
```

```
 Trim() - Likwiduje spacje z prawej i lewej
 strony wyrazu lub ciągu znaków
```

```
}
```

```
end;
```

- Kliknij dwukrotnie na formatce i w wygenerowanym zdarzeniu **OnCreate** wpisz kod:

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

```
var
```

```
 TT: Shortint; // Deklaracja zmiennej
```

```
begin
```

```
{
```

```
 Tu wpisujemy funkcje, które są wykonywane w
 momencie tworzenia formatki.
```

```
}

// Wyczyszczenie edytora w komponencie ComboBox
ComboBox1.Text:= "";
ComboBox2.Text:= "";

// Wyczyszczenie listy komponentu ComboBox
ComboBox1.Items.Clear;
ComboBox2.Items.Clear;

// Wpisanie nazw miesięcy
for TT:= 0 to 11 do
begin
 // Dodanie elementu
 ComboBox1.Items.Add(WyrazDLP(LongMonthNames[1+TT]));
end;

{
 Określa ilość wyświetlanych
 elementów po rozwinięciu listy
}
ComboBox1.DropDownCount:= 12;

// Wpisanie nazw dni
for TT:= 0 to 6 do
begin
 // Dodanie elementu
 ComboBox2.Items.Add(WyrazDLP(LongDayNames[1+TT]));
end;

{
 Zmienne środowiskowe zależne od
 wersji językowej systemu Windows:

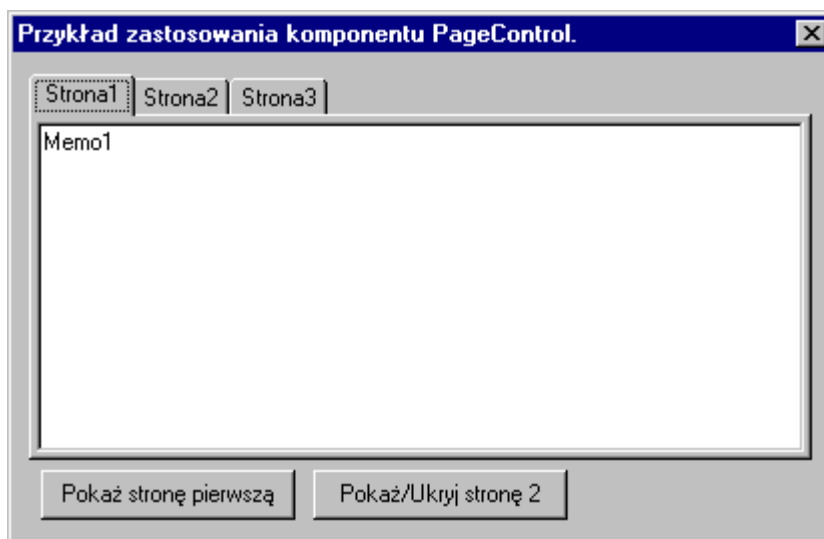
 1) LongMonthNames[Index] - Tablica pełnych nazw miesięcy

 2) LongDayNames[Index] - Tablica pełnych nazw dni
}
end;
```

### Ćwiczenie 5.45. PageControl (Zakładki)


Napisz program, w którym komponenty będą ułożone na różnych zakładkach. Zakładkę drugą będzie można ukryć. Rysunek 5.45.1 przedstawia taki program.

Rysunek 5.45.1





Przykład znajduje się w katalogu Delphi\Cwicz\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**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Win32**);
- Stwórz trzy zakładki wybierając opcję „New Page” (Nowa strona) z menu podręcznego, które wywołuje się prawym klawiszem myszy po zaznaczeniu komponentu **PageControl** znajdującego się na formatce;
- Nadaj tym zakładkom nazwy zgodnie z rysunkiem 5.45.1, przypisując właściwości **Caption** poszczególne nazwy po zaznaczeniu każdej zakładki oddzielnie;
- Na każdej zakładce umieść kilka dowolnych komponentów;
- Wybierz dwa klawisze  (karta **Standard**) i opisz je oraz rozmieść zgodnie z rysunkiem 5.45.1;
- 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;
```



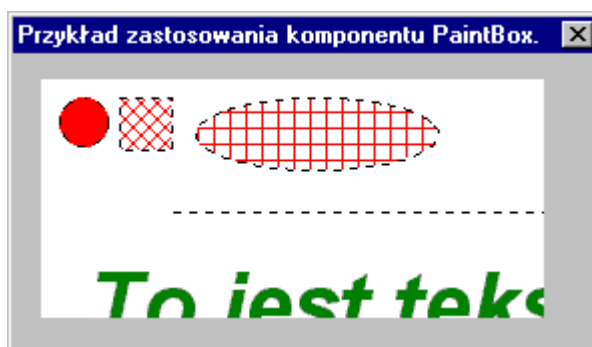
- Kliknij dwukrotnie na klawisz z napisem „Pokaż/Ukryj stronę 2” i w wygenerowanej procedurze wpisz kod:

```
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;
```

### Ćwiczenie 5.46. Rysowanie na ograniczonym obszarze

Wykonaj takie same rysunki jak w ćwiczeniu 5.36. na obszarze ograniczonym. Rysunek 5.46.1 przedstawia taki program.

Rysunek 5.46.1




Przykład znajduje się w katalogu Delphi\Cwicz\PaintBox.

#### Opis komponentu:



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

#### Sposób wykonania:

- Wybierz dwa komponenty **PaintBox**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **System**);
- Zaznacz komponent **PaintBox**, który jest na formatce;
- Przejdź do okna **Object Inspector** (Inspektor Obiektów) i 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;

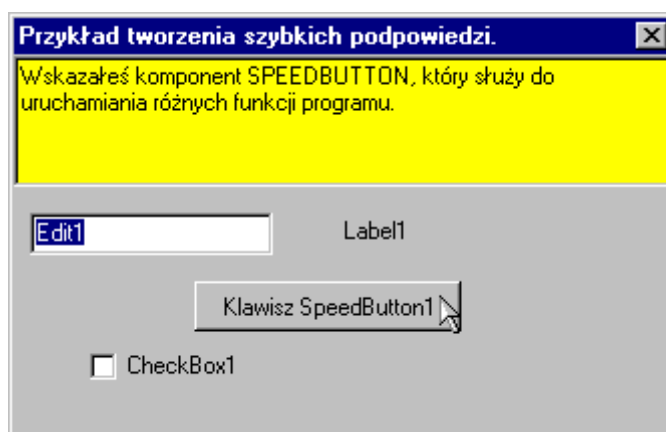
// Wypisanie tekstu na ekranie
TextOut(23, 88, 'To jest tekst.');
```

**end;**  
**end;**

### Ćwiczenie 5.47. Memo do wyświetlania krótkiej pomocy

Wykonaj program, w którym najeżdżenie na element znajdujący się na formacie spowoduje wyświetlenie opisu dla danego elementu. Rysunek 5.47.1 przedstawia przykładowy program.

Rysunek 5.47.1



Przykład znajduje się w katalogu Delphi\Ciwc\Qhelp.

### Sposób wykonania:

- Wybierz komponent **Memo** (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Napisz funkcję, która będzie wyświetlała informację w komponencie **Memo**:

```
function TForm1.qhSzybkaPodpowiedz(txtString: String; okVisible: Boolean): Shortint;
{
 Funkcja wyświetla szybką pomoc opisową w
 komponencie Memo zwanym QuickHelp i zwraca
 wartość 1, jeżeli nowa informacja została
 wyświetlona. W innym przypadku funkcja
 zwróci wartość -1.
}
begin
 // Szybka Podpowieź
 qhSzybkaPodpowiedz:= -1;

 {
 Wyczyszczenie spacji (pustych znaków) z
 prawej i lewej strony tekstu

 Trim() - Likwiduje puste znaki
 z dwóch stron tekstu
 }
 txtString:= Trim(txtString);

 // Ukrycie lub pokazanie komponentu na ekranie
 QuickHelp.Visible:= okVisible;

 {
 Wyłączenie możliwości przejścia do
 tego komponentu za pomocą klawisza TAB
 }
```

```
}
QuickHelp.TabStop:= FALSE;

// Wyłączenie możliwości edycji
QuickHelp.ReadOnly:= TRUE;

{
Zwijanie tekstu w komponencie w przypadku, gdy
tekst wychodzi poza obramowanie
}
QuickHelp.WordWrap:= TRUE;

// Określenie koloru tła
QuickHelp.Color:= clYellow;

// Określenie koloru liter
QuickHelp.Font.Color:= clBlack;

{
Jeżeli zawartość zmiennej "txtString" jest różna od
zawartości komponentu "Trim(QuickHelp.Text)" to
spełniony jest warunek i nowa informacja zostanie
wyświetlona. W innym przypadku informacja nie
będzie zmieniona.
}
if (txtString<>Trim(QuickHelp.Text)) then
begin

 // Wyczyszczenie zawartości komponentu
 QuickHelp.Lines.Clear;

 // Wprowadzenie (dodanie) nowej informacji
 QuickHelp.Lines.Add(txtString);

 {
 Funkcja zwraca wartość 1, gdy
 warunek jest spełniony
 }
 qhSzybkaPodpowiedz:= 1;
end;
end;
```

➤ Zadeklaruj funkcję w typie obiektowym:

**type**

```
TForm1 = class(TForm)
 QuickHelp: TMemo;
```

```
.....
```


```
Label1: TLabel;
```

```
function qhSzybkaPodpowiedz(txtString: String; okVisible: Boolean): Shortint;
```

```

.....
.....
procedure Edit1MouseMove(Sender: TObject; Shift: TShiftState; X,
procedure Edit1Enter(Sender: TObject);
private
 { Private declarations }
public
 { Public declarations }
end;

```

- Zaznacz komponent **Edit**  i przejdź do okna **Object Inspector** (Inspektor Obiektów);
- Wybierz zakładkę **Events** (Zdarzenia) i wybierz zdarzenie **OnEnter**;
- Kliknij dwukrotnie na pustym polu (obok z prawej strony napisu **OnEnter**) i w wygenerowanej procedurze wpisz kod, który będzie wyświetlany w momencie przejścia na komponent **Edit** klawiszem TAB:

```

procedure TForm1.Edit1Enter(Sender: TObject);
begin
 {
 Wyświetla informację o komponencie Edit1, w
 przypadku przejścia na ten komponent za
 pomocą klawisza TAB
 }
 qhSzybkaPodpowiedz('Zaznaczyłeś komponent EDIT, który '+'
 'służy do edycji tekstu.', TRUE);
end;

```

- Wybierz zdarzenie **OnMouseMove** i kliknij dwukrotnie na pustym polu (znajduje się z prawej strony obok napisu **OnMouseMove**). Następnie w wygenerowanej procedurze wpisz kod, który będzie wyświetlany w momencie najechania kursorem myszy na komponent **Edit**:

```

procedure TForm1.Edit1MouseMove(Sender: TObject; Shift: TShiftState; X,
 Y: Integer);
begin
 // Wyświetla informację o komponencie Edit1
 qhSzybkaPodpowiedz('Zaznaczyłeś komponent EDIT, który '+'
 'służy do edycji tekstu.', TRUE);
end;

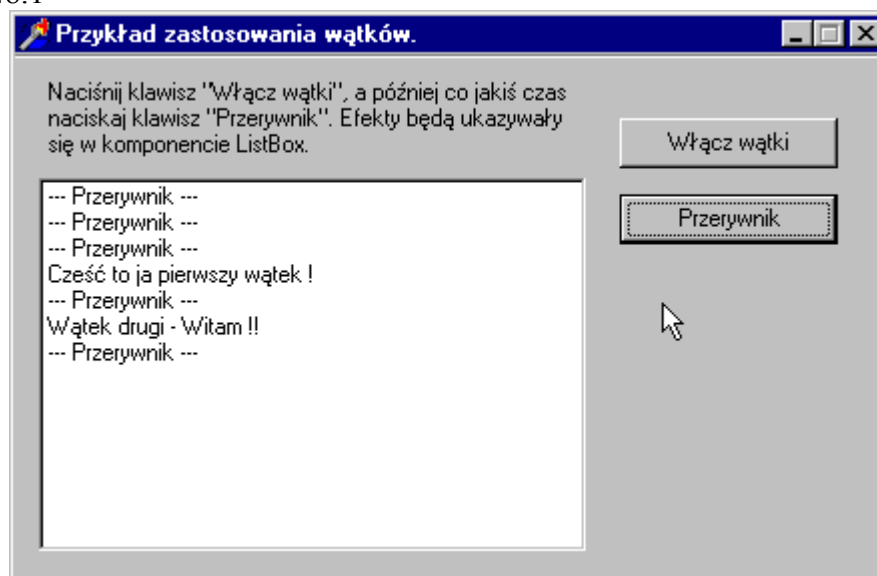
```

- Z następnymi komponentami postępuj tak samo.

### Ćwiczenie 5.48. Wątki

Napisz program, który wykona dwa różne zadania w tym samym czasie co Ty będziesz naciskał na klawisz z napisem „Przerywnik”. Rysunek 5.48.1 przedstawia przykładowy program.

Rysunek 5.48.1



Przykład znajduje się w katalog Delphi\Inne\Thread.

### Opis:

Zastosowanie wątków jest wygodnym rozwiązaniem pozwalającym na wykonywanie zadań przez program w tle z jednoczesnym komunikowaniem się z użytkownikiem.

### Sposób wykonania:

- W celu stworzenia klasy wątku należy wykonać następujące czynności:
  - ❖ Wybierz „New” (Nowy) z menu „File” (Plik);
  - ❖ Będąc na zakładce „New” (Nowy) wybierz „Thread Object” (Klasa wątku);
  - ❖ W ukazanym oknie „New Thread Object” (Nowy wątek) w okienku „Class Name” (Nazwa klasy) wpisz nazwę nowego wątku, np. „Watek\_1”;
  - ❖ Naciśnij klawisz Enter, co spowoduje wygenerowanie nowej klasy wątku:

```
unit Uthread1;
```

```
interface
```

```
uses
```

```
Classes;
```

```
type
```

```
Watek_1 = class(TThread)
```

```
private
```

```
{ Private declarations }
```

```
protected
```

```
procedure Execute; override;
```

```
end;
```

```
implementation
```

```
{ Important: Methods and properties of objects in VCL can only be used in a
```

*method called using Synchronize, for example,*

*Synchronize(UpdateCaption);*




*and UpdateCaption could look like,*

```
procedure Watek_1.UpdateCaption;
begin
 Form1.Caption := 'Updated in a thread';
end; }
```

*{ Watek\_1 }*

```
procedure Watek_1.Execute;
begin
 { Place thread code here }
end;
```

**end.**

- W celu stworzenia drugiej klasy wątku postępuj tak samo jak wyżej;
- Wybierz komponent **Label**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- We właściwości **Caption** komponentu **Label** wpisz tekst zgodny z rysunkiem 5.48.1;
- Wybierz dwa klawisze **Button**  (karta **Standard**) i opisz je zgodnie z rysunkiem 5.48.1;
- Wybierz komponent **ListBox**  (karta **Standard**);
- Ułóż komponenty na formatce zgodnie z rysunkiem 5.48.1;
- Kliknij dwukrotnie na formatce i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 {
 Tu wpisujemy instrukcje, które są
 wykonywane w momencie tworzenia formatki.
 }

 // Wyczyszczenie zawartości komponentu ListBox
 ListBox1.Items.Clear;
end;
```

- Uzupełnij wątek o nazwie „Watek\_1” wpisując kod do szkieletowej postaci stworzonej klasy wątku – wygląd pełnego kodu może wyglądać tak:

```
unit Uthread1;
{
 Wątek pierwszy, który będzie wysyłał
 tekst "Cześć to ja pierwszy wątek" do ListBox1
}
```



**interface****uses**

Classes;

**type**Watek\_1 = **class**(TThread)**private***{ Private declarations }*txtString: **String**;**protected****procedure** UpdateCaption;**procedure** Execute; **override**;**end**;**implementation****uses** SysUtils, Uthread;*// Uthread - Nazwa formatki**{ Important: Methods and properties of objects in VCL can only be used in a method called using Synchronize, for example,**Synchronize(UpdateCaption);**and UpdateCaption could look like,**procedure Watek\_1.UpdateCaption;**begin**Form1.Caption := 'Updated in a thread';**end; }**{ Watek\_1 }***procedure** Watek\_1.UpdateCaption;**begin**

Form1.ListBox1.Items.Add(txtString);

{

*Zapewnia wyświetlenie wyniku w komponencie ListBox, który znajduje się na formatce Form1.*

}

**end**;**procedure** Watek\_1.Execute;**var**

TT: Integer;

**begin***{ Place thread code here }*

{

*Włączenie automatycznego zwolnienia obiektu*

*po jego zakończeniu.*

*Ustawienie właściwości "FreeOnTerminate" na TRUE*

*powoduje automatyczne wywołanie metody*

*Free(), która jest związana z danym wątkiem*

}

FreeOnTerminate:= TRUE;

*// Pętla wykonywana jest 222222 razy*

**for** TT:= 0 to 222222 **do**

**begin**

**if** Terminated **then** Break;

{

*if Terminated then Break;*

*Sprawdzenie wartości właściwości "Terminated".*

*Jeżeli właściwość ta ma wartość TRUE to*

*następuje zakończenie wątku.*

}

txtString:= ";

txtString:= 'Cześć to ja pierwszy wątek !';

**end;**

Synchronize(UpdateCaption);

**end;**

**end.**

- Z uzupełnieniem drugiej klasy wątku postępuj tak samo;
- Zaznacz klawisz z napisem „Włącz wątki” i kliknij na nim dwukrotnie oraz wpisz kod w wygenerowanej procedurze:

**procedure** TForm1.bWatkiClick(Sender: TObject);

**var**

*// Deklaracja obiektu Watek\_1*

WatekPierwszy: Watek\_1;

*// Deklaracja obiektu Watek\_2*

WatekDrugi: Watek\_2;

**begin**

*// Uruchomienie wątku 1*

WatekPierwszy:= Watek\_1.Create(FALSE);

{

*O automatycznym uruchomieniu wątku decyduje jeden parametr konstruktora*

*Watek\_1.Create(FALSE), a mianowicie wartość FALSE.*

*Wartość TRUE powoduje zawieszenie wątku i uruchomienie tego wątku nastąpi po wywołaniu Watek\_1.Resume().*

}

*// Uruchomienie wątku 2*

```
WatekDrugi:= Watek_2.Create(FALSE);
end;
```

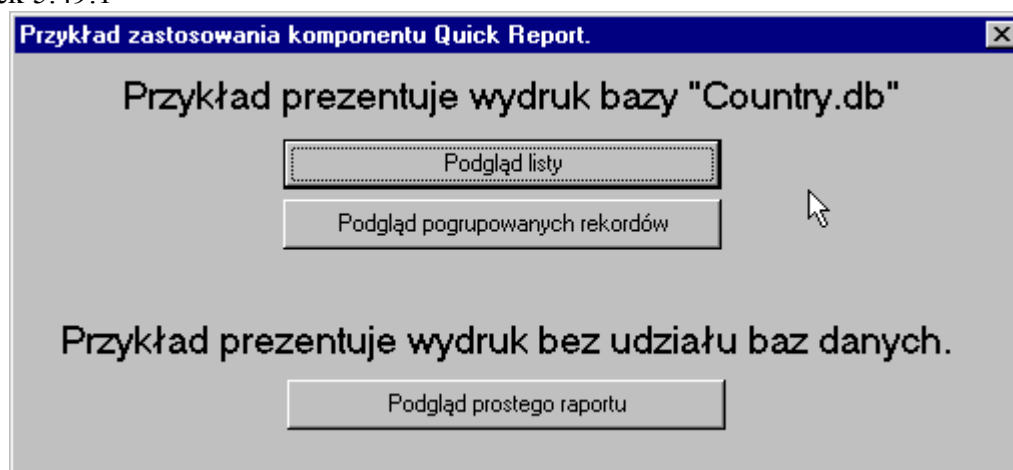
- Zaznacz klawisz z napisem „Przerywnik” i kliknij na nim dwukrotnie oraz wpisz kod w wygenerowanej procedurze:

```
procedure TForm1.bPrzerywnikClick(Sender: TObject);
begin
 // Wyświetlenie komunikatu "--- Przerywnik ---"
 ListBox1.Items.Add('--- Przerywnik ---');
end;
```

### Ćwiczenie 5.49. Wydruki

Napisz program, który będzie umożliwiał trzy rodzaje wydruków:- wydruk prosty, wydruk tabeli; wydruk tabeli pogrupowanej w grupy według alfabetu. Rysunek 5.49.1 przedstawia taki program.

Rysunek 5.49.1





Przykład znajduje się w katalogu Delphi\Cwicz\QReport.

#### Opis komponentu:



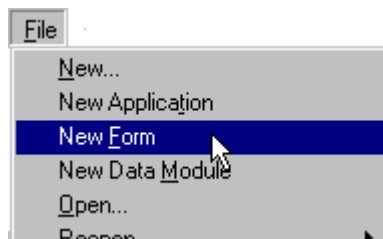
Komponentów służących do tworzenia wydruku jest kilkanaście i znajdują się na zakładce **QReport**. Za ich pomocą można tworzyć proste jak i złożone wydruki graficzne.

#### Sposób wykonania:

- Wybierz trzy komponenty **Label**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz trzy komponenty **Button**  (karta **Standard**);
- Rozmieść i opisz wybrane komponenty zgodnie z rysunkiem 5.49.1;

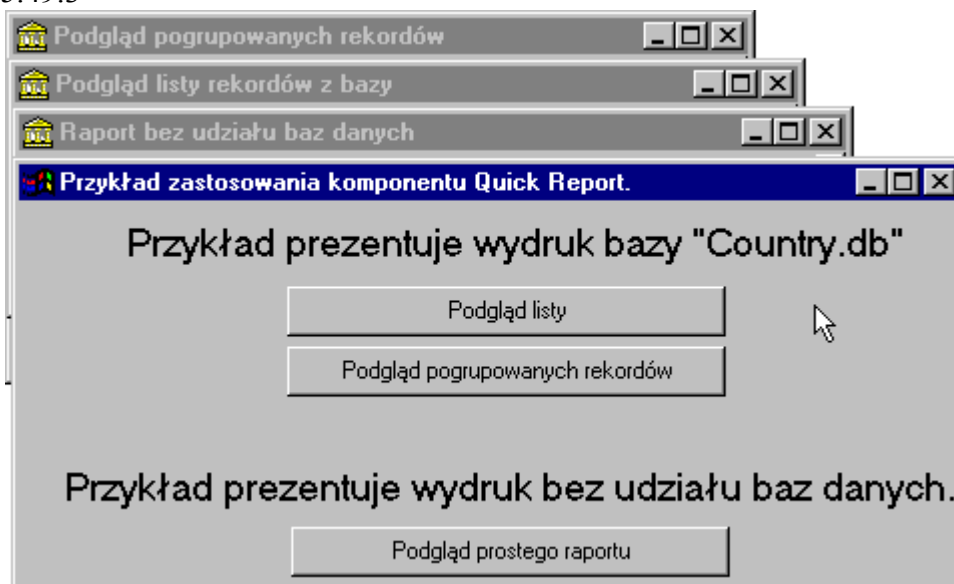
- Pozostaw nazwę głównej formy (domyślną nazwą głównej formy jest **Form1**). Nazwa formy znajduje się w właściwość **Name** formatki;
- Wybierz opcję „New Form” (Nowa forma) z menu „File” (Plik) – rysunek 5.49.2;

Rysunek 5.49.2



- Czynność poprzednią wykonaj jeszcze dwa razy. Tak, aby mieć formę główną i trzy formy dodatkowe – rysunek 5.49.3;

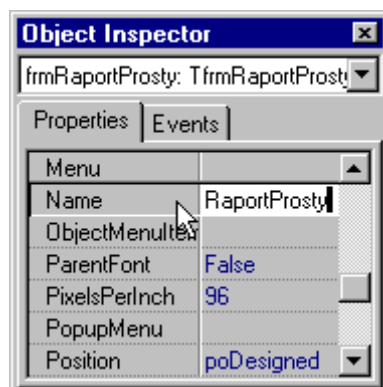
Rysunek 5.49.3



Na trzech formach dodatkowych będą tworzone wydruki.

- Wybraną formę nazwij „frmRaportProsty” wpisując tą nazwę we właściwości **Name** (Nazwa) formatki, uprzednio przechodząc do okna **Object Inspector** (Inspektor Obiektów) – rysunek 5.49.4;

Rysunek 5.49.4

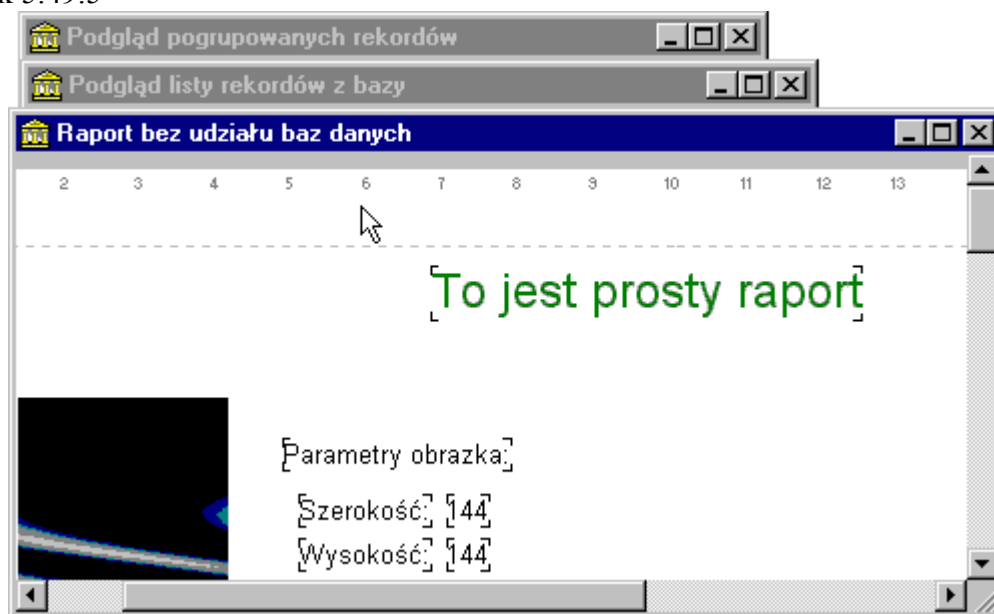


- Tak samo postępuj z następnymi dwoma formami nadając im nazwy „frmListaRekordow” i „frmPogrupowaneRekordy”;

#### Wydruk graficzny:

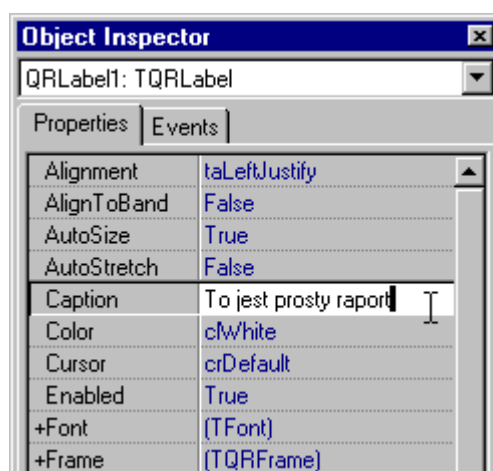
- Przesuń na plan pierwszy formę nazwaną „frmRaportProsty” – rysunek 5.49.5;


Rysunek 5.49.5



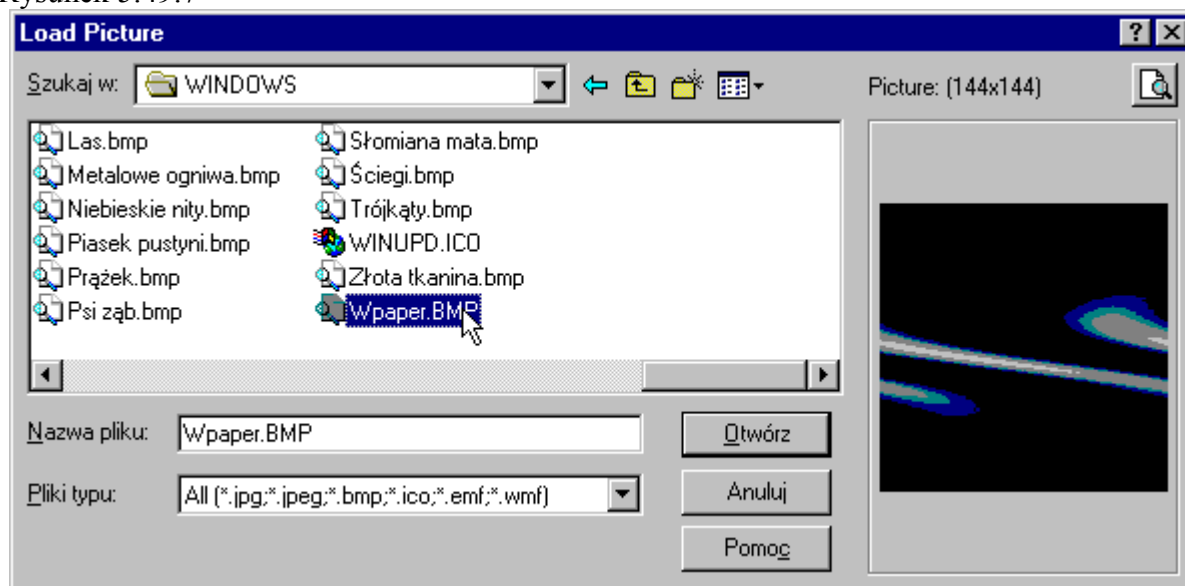
- Wybierz komponent **QuickRep** (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **QReport**);
- Wybierz komponent **QRLabel** (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formatce;
- We właściwości **Caption** komponentu **QRLabel** wpisz dowolny tekst (np. „To jest prosty raport”), uprzednio przechodząc do okna **Object Inspector** (Inspektor Obiektów) – rysunek 5.49.6;


Rysunek 5.49.6



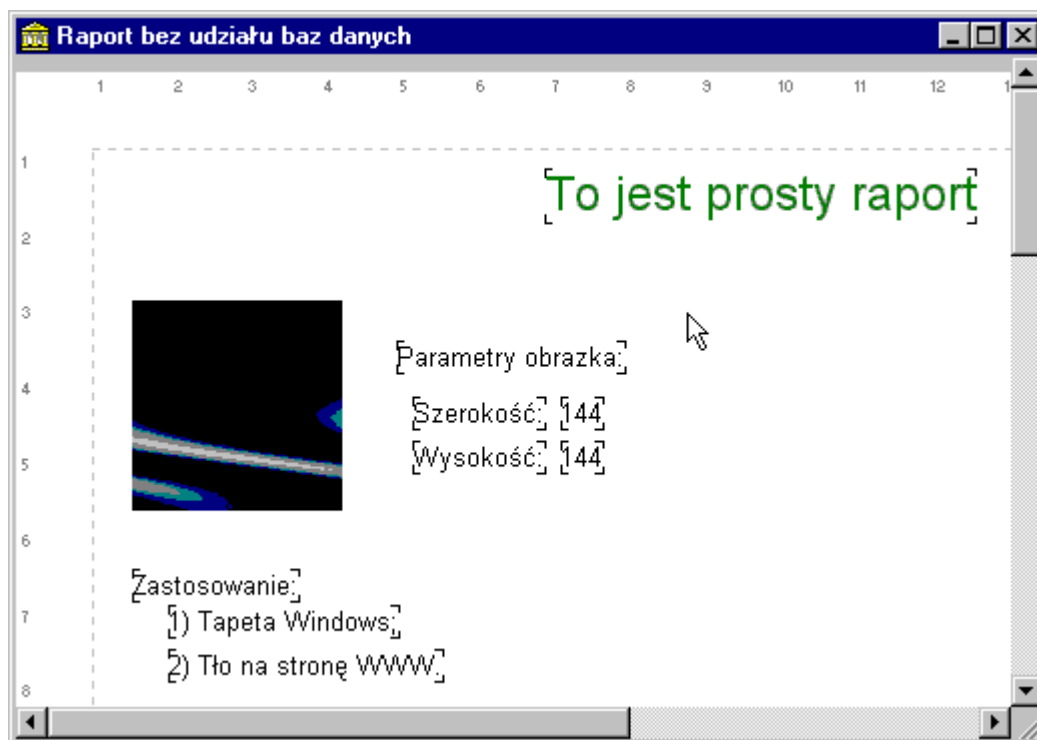
- Wybierz komponent **QRImage**  (karta **QReport**) i kliknij na komponencie **QuickRep1**, który znajduje się na formatce;
- Kliknij dwukrotnie na komponent **QRImage**, co spowoduje otwarcie okna „Picture editor” (Edytor obrazu);
- W oknie tym kliknij na klawisz z napisem „Load” (Wczytaj), co spowoduje otwarcie okna wyboru rysunków – rysunek 5.49.7;

Rysunek 5.49.7



- Wybierz dowolny rysunek i potwierdź wybór klikając na klawisz z napisem „Otwórz”;
- Wybierz kilka komponentów **QRLabel**  **A** (karta **QReport**) i umieść je na komponencie **QuickRep1**, który znajduje się na formatce. UWAGA: Komponenty znajdujące się na komponencie **QuickRep1** muszą przesuwać się razem z tym komponentem, w przypadku przesuwania;
- Wybrane komponenty rozmieść i opisz zgodnie z rysunkiem 5.49.8;

Rysunek 5.49.8



- Wybierz przycisk z napisem „Podgląd prostego wydruku”, który znajduje się na głównej formie i kliknij na nim oraz wpisz kod w wygenerowanej procedurze:

```

procedure TForm1.Button3Click(Sender: TObject);
begin
 // Podgląd prostego raportu
 frmRaportProsty.QuickRep1.Preview;
end;

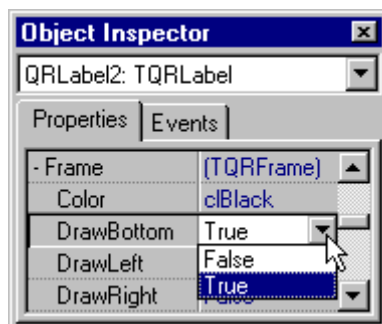
```

#### Wydruk rekordów z bazy w formie tabeli:

- Przesuń na plan pierwszy formę nazwaną „frmListaRekordow”;
- Wybierz komponent **QuickRep** (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **QReport**);
- Wybierz komponent **QRBand** (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formie;
- Wybierz komponent **QRLabel** (karta **QReport**) i umieść go na komponencie **QRBand** oznaczonym jako **rbTitle** we właściwości **BandType** oraz opisz zgodnie z rysunkiem 5.49.10;
- Wybierz komponent **QRBand** (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formie;
- Ustaw właściwość **BandType** komponentu **QRBand** na **rbColumnHeader**, będąc w oknie **Object Inspector** (Inspektora Obiektów);
- Wybierz dwa komponenty **QRLabel** (karta **QReport**) i umieść je na komponencie **QRBand** oznaczonym jako **rbColumnHeader** we właściwości **BandType**;

- Ustaw dla komponentów **QRLabel** właściwość **DrawBottom** na wartość **TRUE** właściwości **Frame** – rysunek 5.49.9;

Rysunek 5.49.9



- Ułóż je obok siebie i opisz je zgodnie z rysunkiem 5.49.10;
- Wybierz komponent **QRBand** (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formie;
- Ustaw właściwość **BandType** komponentu **QRBand** na **rbDetail**, będąc w oknie **Object Inspector** (Inspektora Obiektów);
- Wybierz dwa komponenty **QRDBText** (karta **QReport**) i umieść je na komponencie **QRBand** oznaczonym jako **rbDetail** we właściwości **BandType** zgodnie z rysunkiem 5.49.10;
- Wpisz pierwszemu komponentowi **QRDBText1** nazwę kolumny „Name”, zgodnie z nazwą kolumny w bazie danych we właściwości **DataField**;
- Wpisz drugiemu komponentowi **QRDBText2** nazwę kolumny „Capital”, zgodnie z nazwą kolumny w bazie danych we właściwości **DataField**;
- Wybierz komponent **DataSource** oraz **Query** (obydwa komponenty znajdują się na karcie **Data Access**) i umieść je na formatce;
- Połącz komponent **DataSource** z komponentem **Query** z pomocą właściwości **DataSet** komponentu **DataSource**;
- Połącz komponent **QuickRep1** z komponentem **Query** z pomocą właściwości **DataSet** komponentu **QuickRep1**;
- Połącz komponenty **QRDBText** z komponentem **Query** z pomocą właściwości **DataSet** komponentu **QRDBText**;
- Wybierz komponent **QRBand** (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formie;
- Ustaw właściwość **BandType** komponentu **QRBand** na **rbPageFooter**, będąc w oknie **Object Inspector** (Inspektora Obiektów);
- Wybierz komponent **QRSysData** (karta **QReport**) i umieść go na komponencie **QRBand** oznaczonym jako **rbPageFooter** we właściwości **BandType**;
- Właściwość **Data** komponentu **QRSysData** ustaw na **qrsPageNumber**;

Rysunek 5.49.10





- Napisz procedurę odczytującą rekordy z bazy:

```
procedure TfrmListaRekordow.QReportWydruk;
```

```
begin
```

```
 // QReportWydruk
```

```
 {
```

```
 Wpisanie tekstu, który będzie
 wyświetlany obok numeru strony
```

```
 }
```

```
 QRSysData1.Text:= 'Strona: ';
```

```
 // Zapytanie SQL
```

```
 Query1.Close; // Zamknięcie bazy danych
```

```
 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 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".
```

```
 ORDER BY - klauzula ta służy do sortowania według wybranej kolumny.
```

```
 W naszym przykładzie jest to kolumna "Name".
```

```

}

Query1.Open; // Otwarcie bazy danej
QuickRep1.Preview; // Wyświetlenie podglądu
end;

```

- Zadeklaruj funkcję w typie obiektowym w sekcji **public** (Publicznej), aby była możliwość odwołania się do tej funkcji z modułu głównego:

```

type
 TfrmListaRekordow = class(TForm)
 QuickRep1: TQuickRep;

 QRDBText2: TQRDBText;
 private
 { Private declarations }
 public
 { Public declarations }
 procedure QReportWydruk;
 end;

```






- Wybierz przycisk z napisem „Podgląd listy”, który znajduje się na głównej formie i kliknij na nim oraz wpisz kod w wygenerowanej procedurze:

```

procedure TForm1.Button1Click(Sender: TObject);
begin
 // Podgląd listy rekordów
 frmListaRekordow.QReportWydruk;
end;

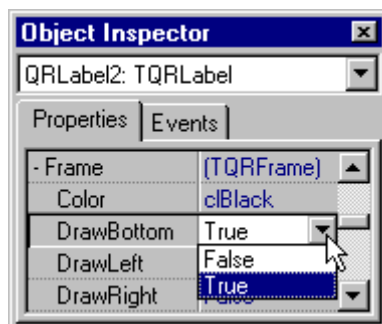
```

#### Wydruk rekordów z bazy w formie pogrupowanej tabeli:

- Przesuń na plan pierwszy formę nazwaną „frmPogrupowaneRekordy”;
- Wybierz komponent **QuickRep**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **QReport**);
- Wybierz komponent **QRBand**  (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formie;
- Wybierz komponent **QRLabel**  **A** (karta **QReport**) i umieść go na komponencie **QRBand** oznaczonym jako **rbTitle** we właściwości **BandType** oraz opisz zgodnie z rysunkiem 5.49.14;
- Wybierz komponent **QRBand**  (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formie;
- Ustaw właściwość **BandType** komponentu **QRBand** na **rbColumnHeader**, będąc w oknie **Object Inspector** (Inspektora Obiektów);
- Wybierz dwa komponent **QRLabel**  **A** (karta **QReport**) i umieść je na komponencie **QRBand** oznaczonym jako **rbColumnHeader** we właściwości **BandType**;

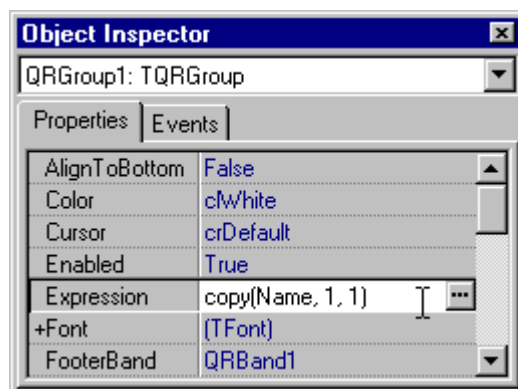
- Ustaw dla komponentów **QRLabel** właściwość **DrawBottom** na wartość **TRUE** właściwości **Frame** – rysunek 5.49.11;

Rysunek 5.49.11



- Ułóż je obok siebie i opisz je zgodnie z rysunkiem 5.49.14;
- Wybierz komponent **QRGroup** (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formie;
- Wpisz we właściwości **Expression** komponentu **QRGroup** oznaczonego jako **rbGroupHeader** wyrażenie „copy(Name, 1, 1)” – rysunek 5.49.12;

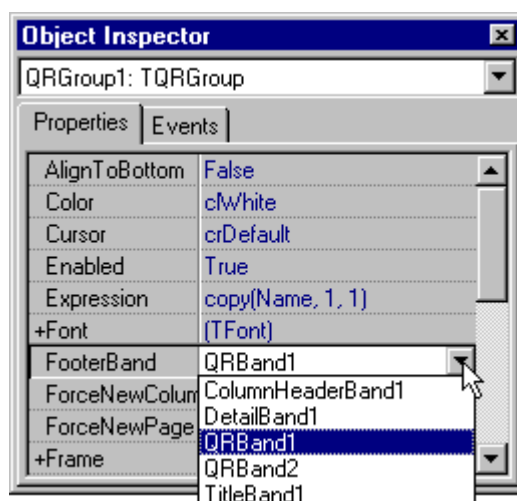
Rysunek 5.49.12










- Wybierz komponent **QRBand** (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formie;
- Ustaw właściwość **BandType** komponentu **QRBand** na **rbGroupFooter**, będąc w oknie **Object Inspector** (Inspektora Obiektów);
- Wybierz komponent **QRLabel** (karta **QReport**) i umieść go na komponencie **QRBand** oznaczonym jako **rbGroupFooter** we właściwości **BandType** oraz opisz zgodnie z rysunkiem 5.49.14;
- Wybierz komponent **QRExpr** (karta **QReport**) i umieść go na komponencie **QRBand** oznaczonym jako **rbGroupFooter** we właściwości **BandType** zgodnie z rysunkiem 5.49.14;
- Wpisz we właściwości **Expression** komponentu **QRExpr** wyrażenie „sum(Money)”;
- Właściwość **ResetAfterPrint** komponentu **QRExpr** ustaw na **TRUE**, co zapobiegnie sumowaniu się obliczeń poszczególnych grup;
- Ustaw właściwość **FooterBand** komponentu **QRGroup** oznaczonego jako **rbGroupHeader** na **QRBand** (na tym komponencie kończy się grupa i taki

komponent oznaczony jest jako **rbGroupFooter** we właściwości **BandType**) – rysunek 5.49.13;

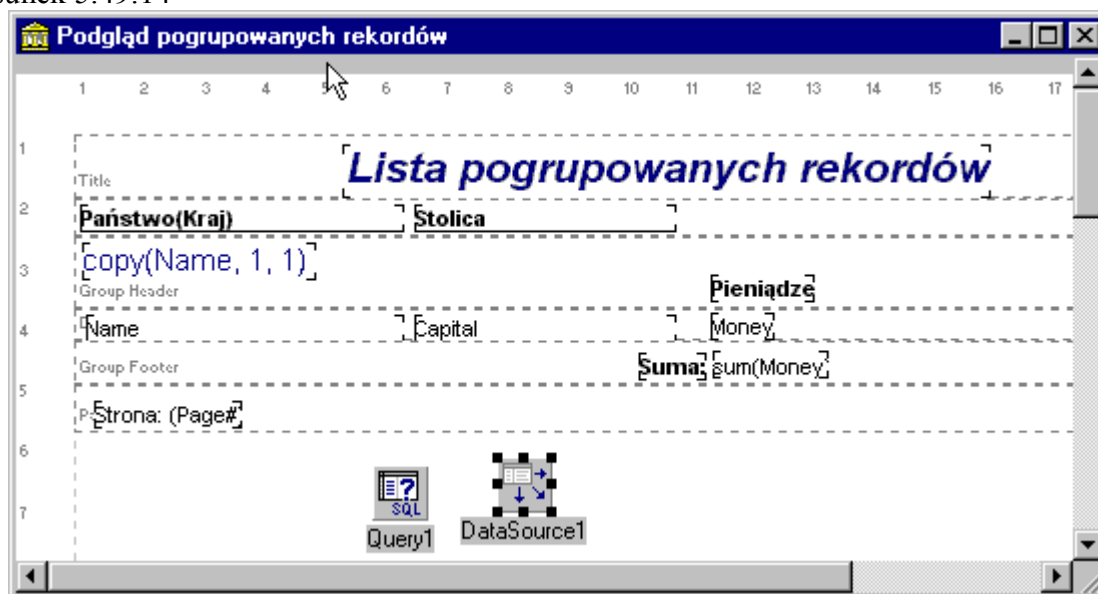
Rysunek 5.49.13



- Wybierz komponent **QRExp**  (karta **QReport**) i umieść go na komponencie **QRGroup**, który jest oznaczony jako **rbGroupHeader**;
- Wpisz we właściwości **Expression** komponentu **QRExp** wyrażenie „copy(Name, 1, 1)”;
- Ustaw właściwość **Master** komponentu **QRGroup** oznaczonego jako **rbGroupHeader** we właściwości **BandType** na **QuickRep1**;
- Wybierz komponent **QRBand**  (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formie;
- Ustaw właściwość **BandType** komponentu **QRBand** na **rbDetail**, będąc w oknie **Object Inspector** (Inspektora Obiektów);
- Wybierz trzy komponenty **QRDBText**  **A** (karta **QReport**) i umieść je na komponencie **QRBand** oznaczonym jako **rbDetail** we właściwość **BandType** zgodnie z rysunkiem 5.49.14;
- Wpisz pierwszemu komponentowi **QRDBText1** nazwę kolumny „Name”, zgodnie z nazwą kolumny w bazie danych we właściwości **DataField**;
- Wpisz drugiemu komponentowi **QRDBText2** nazwę kolumny „Capital”, zgodnie z nazwą kolumny w bazie danych we właściwości **DataField**;
- Wpisz trzeciemu komponentowi **QRDBText3** nazwę kolumny „Money”, zgodnie z nazwą kolumny w bazie danych we właściwości **DataField**;
- Wybierz komponent **QRBand**  (karta **QReport**) i umieść go na komponencie **QuickRep1**, który znajduje się na formie;
- Ustaw właściwość **BandType** komponentu **QRBand** na **rbPageFooter**, będąc w oknie **Object Inspector** (Inspektora Obiektów);
- Wybierz komponent **QRSysData**  **sys** (karta **QReport**) i umieść go na komponencie **QRBand** oznaczonym jako **rbPageFooter** za pomocą właściwości **BandType**;
- Właściwość **Data** komponentu **QRSysData** ustaw na **qrsPageNumber**;
- Wybierz komponent **DataSource**  oraz **Query**  (obydwa komponenty znajdują się na karcie **Data Access**) i umieść je na formatce;

- Połącz komponent **DataSource** z komponentem **Query** z pomocą właściwości **DataSet** komponentu **DataSource**;
- Połącz komponent **QuickRep1** z komponentem **Query** z pomocą właściwości **DataSet** komponentu **QuickRep1**;
- Połącz komponenty **QRDBText** z komponentem **Query** z pomocą właściwości **DataSet** komponentu **QRDBText**;

Rysunek 5.49.14



- Napisz procedurę odczytującą rekordy z bazy:

```

procedure TfrmPogrupowaneRekordy.QReportWydruk;
begin
 // QReportWydruk

 {
 Wpisanie tekstu, który będzie
 wyświetlany obok numeru strony
 }
 QRSysData1.Text:= 'Strona: ';

 // Zapytanie SQL
 Query1.Close; // Zamknięcie bazy danej
 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 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".

ORDER BY - klauzula ta służy do sortowania według wybranej kolumny.  
 W naszym przykładzie jest to kolumna "Name".

}

```
Query1.Open; // Otwarcie bazy danej
QuickRep1.Preview; // Wyświetlenie podglądu
end;
```

- Zadeklaruj funkcję w typie obiektowym w sekcji **public** (Publicznej), aby była możliwość odwołania się do tej funkcji z modułu głównego:

```
type
 TfrmPogrupowaneRekordy = class(TForm)
 QuickRep1: TQuickRep;

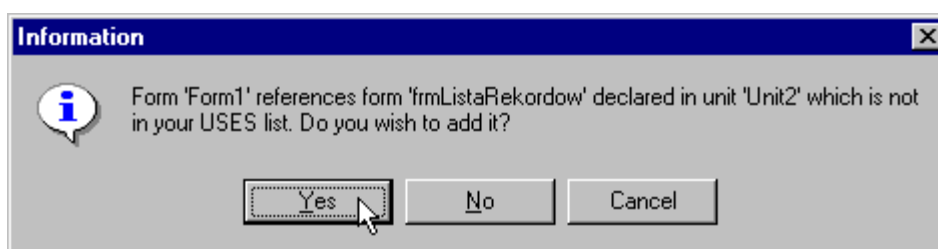
 Query1: TQuery;
 private
 { Private declarations }
 public
 { Public declarations }
 procedure QReportWydruk;
 end;
```

- Wybierz przycisk z napisem „Podgląd pogrupowanych rekordów”, który znajduje się na głównej formie i kliknij na nim oraz wpisz kod w wygenerowanej procedurze:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
 // Podgląd pogrupowanych rekordów
 frmPogrupowaneRekordy.QReportWydruk;;
end;
```

W momencie kompilowania programu Delphi wyświetli komunikat „Form ‘Form1’ references from ‘frmListaRekordow’ declared in unit ‘Unit2’ which is not in your USES list. Do you wish to add it?” (Forma ‘Form1’ wywołuje deklaracje z ‘frmListaRekordow’ znajdującą się w jednostce ‘Unit2’, która nie jest zadeklarowana w liście po słowie USES. Czy zyczysz sobie zadeklarować tę jednostkę ?) – rysunek 5.49.15.

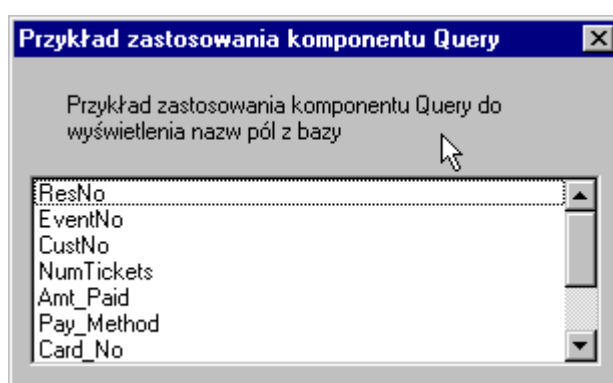
Rysunek 5.49.15



Przez kliknięcie na klawisz z napisem „Yes” (Tak) wyrażamy zgodę na zadeklarowanie jednostki „Unit2”. W podobnych przypadkach postępuj tak samo.

### Ćwiczenie 5.50. Wyświetlenie nazw kolumn z bazy

Napisz program, który będzie umożliwiał odczytanie nazw kolumny występujących w bazie. Rysunek 5.50.1 przedstawia przykładowy program.





Przykład znajduje się w katalogu Delphi\Cwicz\dbfield.

#### Opis komponentu:



**Query** jest komponentem służącym do reprezentowania danych, które są efektem zadanego pytania SQL w stosunku do jednej lub większej ilości tabel.

#### Sposób wykonania:

- Wybierz komponent **Query**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Data Access**);
- Wybierz komponent **ListBox**  (karta **Standard**);
- Kliknij dwukrotnie na formie, 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 bazy danych
Query1.SQL.Clear; // Czyszczenie zapytania SQL
```

```
Query1.SQL.Add('SELECT * FROM Reservat.db');
{
SELECT * FROM Country.db
Wyświetla zawartość tablicy Country.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 danych
```

```
//-- 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)

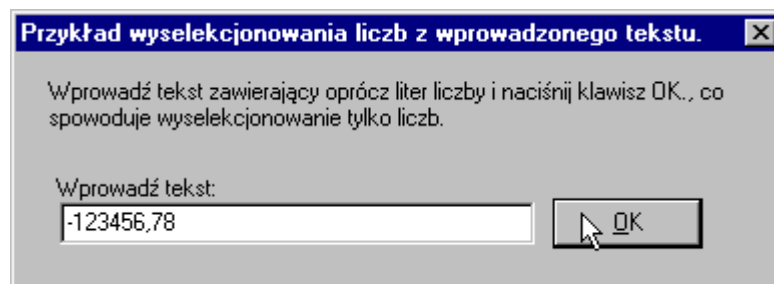
 Query1.Fields[Index].DisplayName
 Wyświetla nazwę pola występującego w bazie o zadanym indeksie
 }
end;
```



### Ćwiczenie 5.51. Wyselekcjonowanie liczb z tekstu



Napisz program, który wyselekcjonuje liczby z wprowadzonego tekstu. Rysunek 5.51.1 przedstawia przykładowy program.

Rysunek 5.51.1



Przykład znajduje się w katalogu Delphi\Inne\Edycja.

#### Sposób wykonania:

- Wybierz komponent **Edit**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz dwa klawisze **Button**  (karta **Standard**);
- Napisz funkcję, która wyselekcjonuje liczby z wprowadzonego tekstu (powyżej zdarzenia, np. **OnCreate** = procedurze **FormCreate**, a poniżej słowa **implementation**):

```
function jbTylkoLiczby(txtString: String; chrComma: Char): String;
```

```
{
 Funkcja z wprowadzonego tekstu
 wybiera 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 "-Przy123kla-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 (znaku 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 wykonywane 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 z 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:

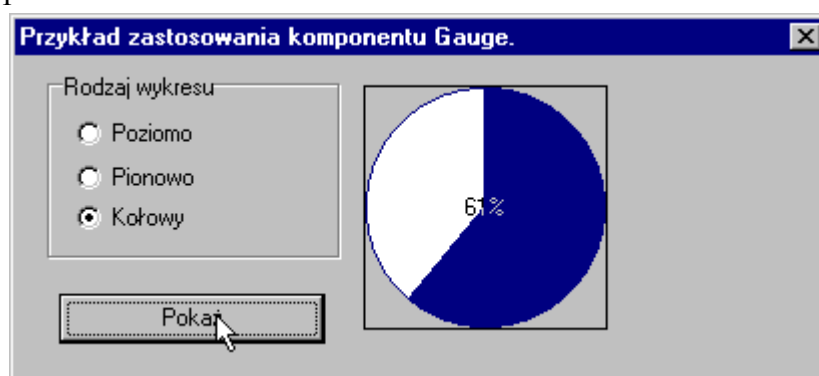
```
procedure TForm1.bOKClick(Sender: TObject);
begin
{
 Wywołuje funkcję do wyselekcjonowania
 liczb z wprowadzonego tekstu

 jbTylkoLiczby(Wprowadź_Tekst,
 Znak_reprezentujący_przecinek);
}
 Edit1.Text:= jbTylkoLiczby(Edit1.Text, ',');
end;
```

### Ćwiczenie 5.52. Gauge


Napisz program, taki sam jak w ćwiczeniu 5.9 wykorzystując komponent **Gauge**. Program ma mieć możliwość zmiany położenia wykresu. Rysunek 5.52 przedstawia przykładowy program.

Rysunek 5.52.1







Przykład znajduje się w katalogu Delphi\Cwicz\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**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) (karta **Standard**);
- Wybierz trzy komponenty **RadioButton**  (karta **Standard**);
- Wybierz komponent **Button**  (karta **Standard**);
- Ułóż i opisz te komponenty zgodnie z rysunkiem 5.52.1;
- 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;
```

- Wybierz **RadioButton** z napisem „Poziomo” i kliknij na nim dwukrotnie oraz 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;
```

- Wybierz **RadioButton** z napisem „Pionowo” i kliknij na nim dwukrotnie oraz 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;
```

- Wybierz **RadioButton** z napisem „Kołowy” i kliknij na nim dwukrotnie oraz 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;
```

- Wybierz klawisz z napisem „Pokaż” i kliknij na nim oraz wpisz kod w wygenerowanej procedurze:

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

```
var
```

```
TT: Integer; // Zadeklarowanie zmiennej "TT"
```

```
begin
```

```
// Pokaż postęp wykonanej procy pętli FOR
```

```
for TT:= 0 to 99 do
```

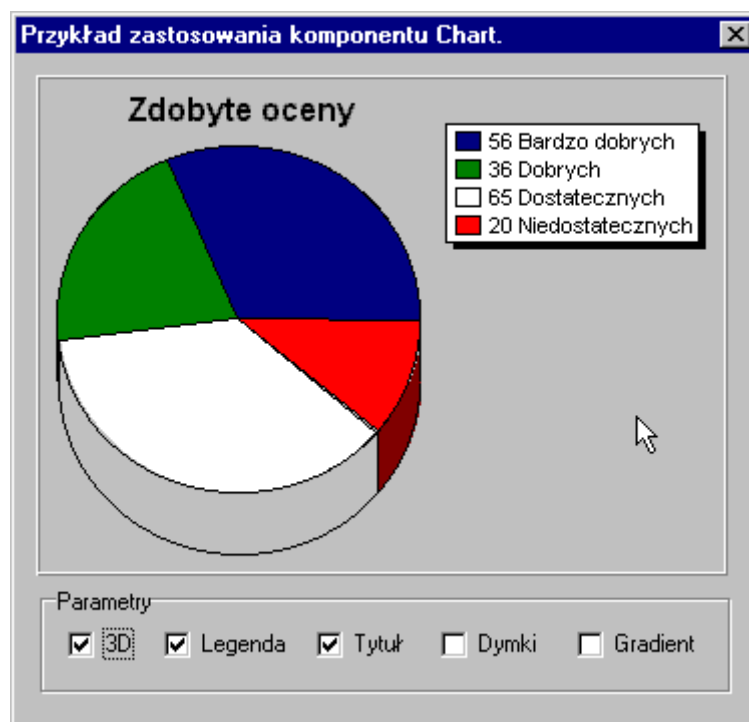
```
Gauge1.Progress:= 1+TT;
```

```
// Pętla FOR zostanie wykonana 1000 razy
end;
```

### Ćwiczenie 5.53. Chart – wykresy


Napisz program, który przedstawi w formie wykresu kołowego np. zdobyte oceny. Rysunek 5.53.1 przedstawia taki program.

Rysunek 5.53.1






Przykład znajduje się w katalogu Delphi\Cwicz\Chart.

#### Opis komponentu:

 **Chart** jest komponentem umożliwiającym tworzenie różnego rodzaju wykresów dwuwymiarowych i trójwymiarowych. Znajduje się on na karcie **Additional** i jest dostępny w Delphi oznaczonym jako Client/Server.

#### Sposób wykonania:

- Wybierz komponent **Chart**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) (karta **Additional**);
- Wybierz komponent **GroupBox**  (karta **Standard**);
- Wybierz kilka komponentów **CheckBox**  (karta **Standard**);
- Poukładaj i opisz te komponenty zgodnie z rysunkiem 5.53.1;
- Kliknij dwukrotnie na formie i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 // Inicjalizacja komponentu Chart
```

*// Wyzerowanie wykresu*

Series1.Clear;

*// Dodanie danych do wykresu*

Series1.Add(56, 'Bardzo dobrych', clNavy);

Series1.Add(36, 'Dobrych', clGreen);

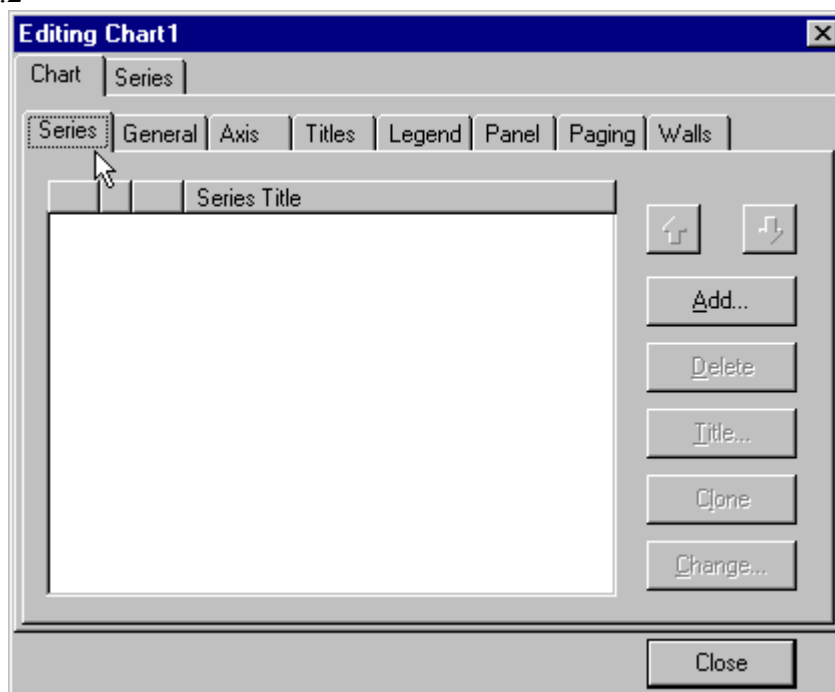
Series1.Add(65, 'Dostatecznych', clWhite);

Series1.Add(20, 'Niedostatecznych', clRed);

**end;**

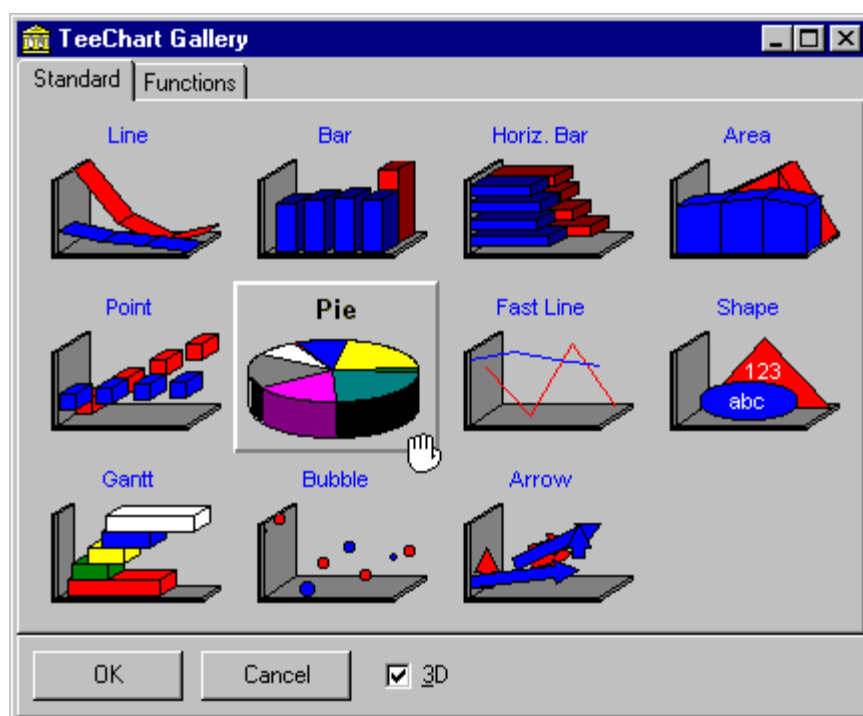
- Zaznacz komponent **Chart** i kliknij na nim dwukrotnie;
- Przejdź do zakładki **Series** (Seria), która znajduje się na zakładce **Chart** (Wykres) - rysunek 5.53.2;

Rysunek 5.53.2



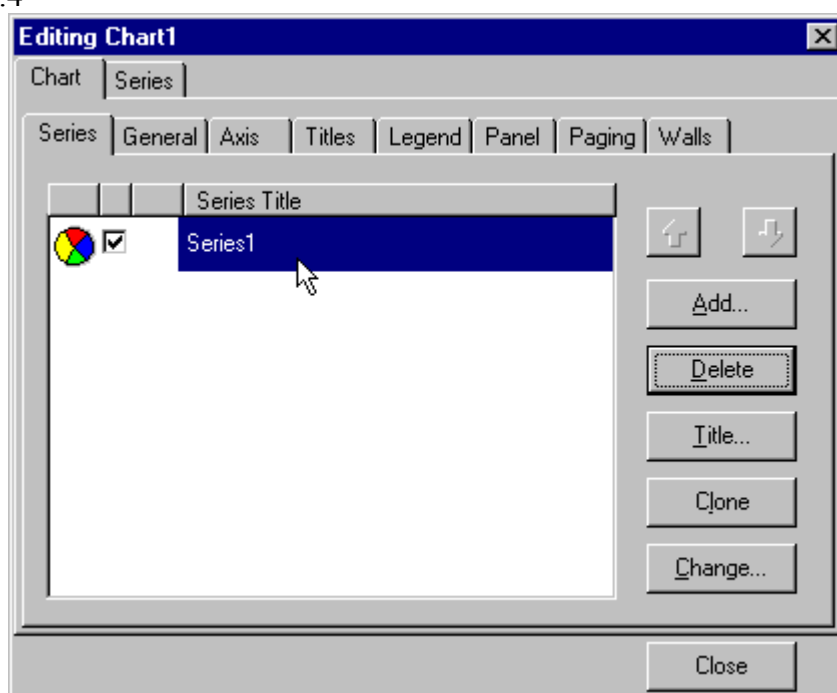
- Kliknij na klawisz z napisem **Add** (Dodaj), co spowoduje ukazanie się okna z rodzajami wykresów do wyboru – rysunek 5.53.3

Rysunek 5.53.3



- Wybierz wykres **Pie** (Kołowy), co spowoduje dodanie tego wykresu do listy – rysunek 5.53.4;

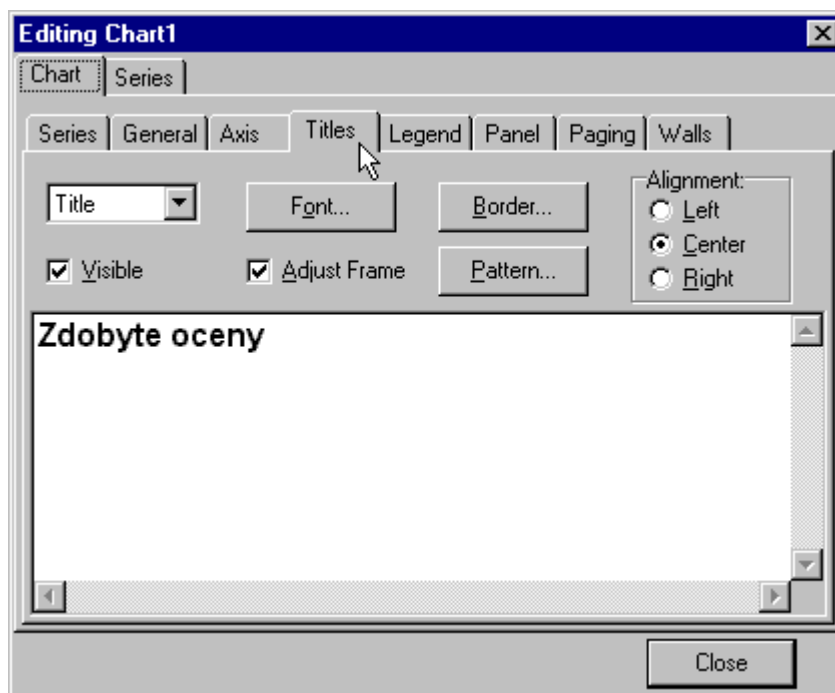
Rysunek 5.53.4



- Przejdź do zakładki **Titles** (Tytuły), która znajduje się na zakładce **Chart** (Wykres) i wpisz tekst „Zdobyte oceny” - rysunek 5.53.5;

Rysunek 5.53.5





- Kliknij dwukrotnie na napis **3D** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
 // Wykres trójwymiarowy/dwuwymiarowy
 Chart1.View3D:= CheckBox1.Checked;
end;
```

- Ustaw właściwość **Checked** na wartość TRUE komponentu z napisem **3D** w oknie Inspektora Obiektów;
- Kliknij dwukrotnie na napisem **Legenda** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.CheckBox2Click(Sender: TObject);
begin
 // Pokazuje/ukrywa Legendę
 Chart1.Legend.Visible:= CheckBox2.Checked;
end;
```

- Ustaw właściwość **Checked** na wartość TRUE komponentu z napisem **Legenda** w oknie Inspektora Obiektów;
- Kliknij dwukrotnie na napis **Tytuł** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.CheckBox3Click(Sender: TObject);
begin
 // Pokazuje/ukrywa Tytuł
 Chart1.Title.Visible:= CheckBox3.Checked;
end;
```

- Ustaw właściwość **Checked** na wartość TRUE komponentu z napisem **Tytuł** w oknie Inspektora Obiektów;
- Kliknij dwukrotnie na napis **Dymki** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.CheckBox4Click(Sender: TObject);
begin
 // Pokazuje/ukrywa Dymki
 Series1.Marks.Visible:= CheckBox4.Checked;
end;
```

- Ustaw właściwość **Checked** na wartość TRUE komponentu z napisem **Dymki** w oknie Inspektora Obiektów;
- Kliknij dwukrotnie na napis **Gradient** i w wygenerowanej procedurze wpisz kod:

```
procedure TForm1.CheckBox5Click(Sender: TObject);
begin
 {
 Włącza/wyłącza gradient
 (przejsie z jednego koloru na inny kolor)
 }
 Chart1.Gradient.Visible:= CheckBox5.Checked;

 // Określenie pierwszego koloru
 Chart1.Gradient.StartColor:= clYellow;

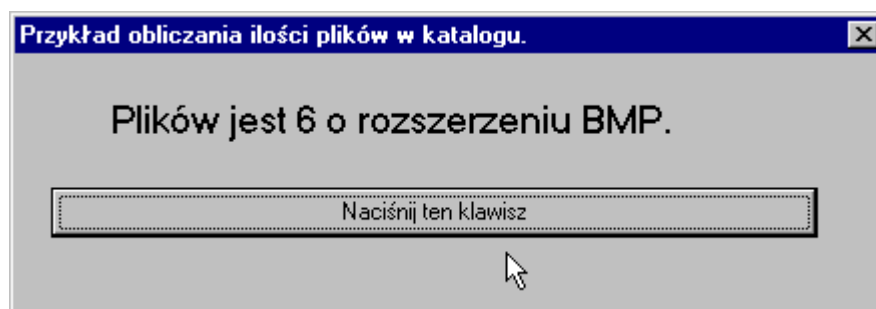
 // Określenie drugiego koloru
 Chart1.Gradient.EndColor:= clRed;

 // Określenie kierunku przechodzenia kolorów
 Chart1.Gradient.Direction:= gdBottomTop;
end;
```

- Ustaw właściwość **Checked** na wartość TRUE komponentu z napisem **Gradient** w oknie Inspektora Obiektów.



### Ćwiczenie 5.54. Liczenie plików

Napisz program, który poda liczbę plików o rozszerzeniu BMP i PAS znajdujących się w podanym katalogu. Rysunek 5.54.1. przedstawia przykładowy program.



Przykład znajduje się w katalogu Delphi\Inne\liczplik.

**Sposób wykonania:**

- Wybierz komponent **Label**  (sposób wybrania i umieszczenia komponentu na formacie opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponent **Button**  z palety komponentów (karta **Standard**);
- Ułóż te komponenty zgodnie z rysunkiem 5.54.1;
- Napisz funkcję obliczającą ilość plików w podanym katalogu (powyżej zdarzenia, np. **OnCreate** = procedurze **FormCreate**, a poniżej słowa **implementation**):

**function** IloscPlikow(txtSciezka, txtRozszerzenie: **String**): Integer;

```
{
Funkcja "IloscPlikow" zwraca liczbę plików znajdujących
się w podanym katalogu (parametr: txtSciezka) i o
podanym rozszerzeniu (parametr: txtRozszerzenie).
Jeżeli nie będzie żadnych plików, to
funkcja zwróci wartość 0.
}
```

**var**

SR: TSearchRec;

```
{
 SR: TSearchRec;
 Zadeklarowanie rekordu zawierającego
 informacje o znalezionym pliku.
}
```

```
// Zadeklarowanie zmiennej liczbowej 'numLicznik'
numLicznik: Integer;
```

```
// Zadeklarowanie zmiennej tekstowej 'txtFullPath'
txtFullPath: String;
```

**begin**

```
// IloscPlikow
```

```
// Przypisanie zmiennej 'txtFullPath'
// parametrów funkcji
txtFullPath:= "";
txtFullPath:= Trim(txtSciezka)+Trim(txtRozszerzenie);
```

```
// Przypisanie zmiennej 'numLicznik' wartości 0
numLicznik:= 0;
```

```
{
 FindFirst(Sciezka, Atrybut, SR - Rekord TSearchRec)
 Funkcja szuka pliku w podanym
 folderze (parametr: Ścieżka), spełniającego
 atrybuty podane w parametrze 'Atrybut'.
 Informacja o pierwszym znalezionym pliku
 jest umieszczana w parametrze SR.
 Funkcja zwróci 0, jeżeli plik zostanie
 znaleziony, w innym przypadku zostanie
```

```
 zwrócony kod błędu.
 }
 if (FindFirst(txtFullPath, faAnyFile, SR) = 0) then
 begin

 repeat
 if (SR.Attr<>faDirectory) then numLicznik:= numLicznik+1;
 {
 if (SR.Attr<>faDirectory) then
 Warunek będzie spełniony, gdy szukany
 element nie będzie katalogiem.

 numLicznik:= numLicznik+1;
 Powiększa zmienną 'numLicznik' o jeden, w
 przypadku spełnienia warunku.
 }
 until(FindNext(SR)<>0);
 {
 FindNext(SR)
 Funkcja kontynuuje szukanie następnego pliku o
 takich samych parametrach, które były podane
 przy wywoływaniu funkcji 'FindFirst'.
 Dane o szukanych plikach są przechowywane w
 parametrze SR. Funkcja zwróci wartość 0 w
 przypadku znalezienia pliku. Jeżeli plik
 nie zostanie znaleziony funkcja zwróci kod błędu.
 }

 FindClose(SR);
 {
 FindClose(SR)
 Funkcja zwalnia pamięć, która została
 przydzielona przez funkcję 'FindFirst'.
 }

 end;

 // Zwraca wartość zmiennej 'numLicznik'
 IloscPlikow:= numLicznik;
 end;
```

- Kliknij dwukrotnie na formatce i w wygenerowanej procedurze **OnCreate** wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 // Wywołanie funkcji 'IloscPlikow'
 // Obliczenie ilości plików o rozszerzeniu BMP
 // Sposób wywołania funkcji IloscPlikow:
 // IloscPlikow(Nazwa_Katalogu, Rozszerzenie_Pliku)
```

```
Label1.Caption:= 'Plików jest '+
 IntToStr(IloscPlikow(", '*.bmp'))+
 ' o rozszerzeniu BMP.';
end;
```

UWAGA: Brak nazwy katalogu oznacza, że pliki będą pobierane z katalogu bieżącego.

➤ Kliknij dwukrotnie na klawisz i w wygenerowanej procedurze **OnClick** wpisz kod:


```
procedure TForm1.Button1Click(Sender: TObject);
begin
 // Wywołanie funkcji 'IloscPlikow'
 // Obliczenie ilości plików o rozszerzeniu PAS
 Button1.Caption:= 'Plików jest '+
 IntToStr(IloscPlikow(", '*.PAS'))+
 ' o rozszerzeniu PAS.';
end;
```

### Ćwiczenie 5.55. Okno InputBox

*Napisz program, który umożliwi wpisanie tekstu przy pomocy polecenia **InputBox**.*

Przykład znajduje się w katalogu Delphi\Cwicz\Inputbox.

#### Sposób wykonania:

- Wybierz komponent **Button**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Kliknij na ten klawisz dwukrotnie i w wygenerowanej procedurze **OnClick** wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 // Wywołanie okna edycyjnego 'InputBox'
 Button1.Caption:= InputBox('Tekst',
 'Wpisz jakiś tekst:',
 'To jest tekst domyślny');

 {
 InputBox(Nazwa_Okna,
 Tekst_Opisujący_Edytorek,
 Tekst_Domyslny);
 }

 InputBox - Funkcja zwraca tekst wprowadzony przez
 użytkownika w przypadku naciśnięcia
 klawisza ENTER lub kliknięcia na
 klawisz z napisem OK.
 W przeciwnym razie zostanie zwrócony
 tekst domyślny wprowadzony w trzecim
 parametrze funkcji 'Tekst_Domyslny'
 (w tym przykładzie będzie to tekst:
```

"To jest tekst domyślny").

*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 okienkiem edycyjnym, np. "Wpisz jakiś tekst:".

*Tekst\_Domyslny* - W tym parametrze wpisujemy tekst, który zostanie wyświetlony w edytorze.

```
}
end;
```

### Ćwiczenie 5.56. Plik rekordowy



Napisz program, który zapisze zawartość tabeli **StringGrid** do pliku oraz umożliwi odczytanie tej tabeli z pliku. Rysunek 5.56.1 przedstawia taki program.

Rysunek 5.56.1.



Przykład znajduje się w katalogu Delphi\Inne\Plikrek.

#### Sposób wykonania:

- Wybierz cztery komponenty **Button**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Umieść je na formatce i opisz zgodnie z rysunkiem 5.56.1;
- Wybierz komponent **StringGrid**  (karta **Additional**) i umieść go na formatce;
- Zadeklaruj stałą tekstową 'prNazwaPliku' poniżej dyrektywy **uses**:

```
const prNazwaPliku = 'baza.dat';
```

- W miejscu deklaracji typów (poniżej **uses** i **const**) wpisz:

```
// Deklaracja pliku rekordowego
type
```

```
PlikRek = record
 Imie: String[20];
 Nazwisko: String[50];
 Miasto: String[140];
end;
```

- W deklaracji zmiennych globalnych modułu (pomiędzy słowami **var** i **implementation**) wpisz deklarację zmiennej plikowej:

```
PR: file of PlikRek; // Deklaracja zmiennej plikowej
```

- Kliknij jednokrotnie na formatce i w wygenerowanej procedurze **OnCreate** wpisz kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 // Opisanie nagłówków poszczególnych kolumn
 StringGrid1.Cells[0, 0] := 'Imię';
 StringGrid1.Cells[1, 0] := 'Nazwisko';
 StringGrid1.Cells[2, 0] := 'Miasto';
end;
```

- Kliknij dwukrotnie na klawiszu z napisem **Wypełnij tabelę** i w wygenerowanej procedurze **OnClick** wpisz kod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 // Wypełnienie kolumn

 //-1-
 StringGrid1.Cells[0, 1] := 'Jan';
 StringGrid1.Cells[1, 1] := 'Biernat';
 StringGrid1.Cells[2, 1] := 'Bielsko';

 //-2-
 StringGrid1.Cells[0, 2] := 'Krzysz';
 StringGrid1.Cells[1, 2] := 'Dalton';
 StringGrid1.Cells[2, 2] := 'Warszawa';

 //-3-
 StringGrid1.Cells[0, 3] := 'Maria';
 StringGrid1.Cells[1, 3] := 'Janosik';
 StringGrid1.Cells[2, 3] := 'Gdańsk';

 //-4-
 StringGrid1.Cells[0, 4] := 'Ela';
 StringGrid1.Cells[1, 4] := 'Perepeczko';
 StringGrid1.Cells[2, 4] := 'Kraków';
end;
```

- Kliknij dwukrotnie na klawisz z napisem **Wyczyść tabelę** i w wygenerowanej procedurze **OnClick** wpisz kod:

```
procedure TForm1.Button2Click(Sender: TObject);
var
 AA, BB: Integer;
begin
 // Czyści tabelę
 for AA:= 0 to StringGrid1.RowCount-1 do
 for BB:= 0 to StringGrid1.ColCount-1 do
 StringGrid1.Cells[BB, 1+AA]:= '';
end;
```

- Kliknij dwukrotnie na klawisz z napisem **Odczytaj tabelę** i w wygenerowanej procedurze **OnClick** wpisz kod:

```
procedure TForm1.Button3Click(Sender: TObject);
// Deklaracja zmiennych
var
 R1: PlikRek; // Typ rekordowy
 TT: Longint;
begin
 // Odczytanie tabeli z pliku rekordowego

 if (FileExists(prNazwaPliku) = TRUE) then
 // Odczytaj plik, jeżeli istnieje.
 begin

 // Powiązanie nazwy pliku z fizycznym plikiem na dysku
 AssignFile(PR, prNazwaPliku);

 Reset(PR); // Otwarcie istniejącego pliku

 // Odczyt tabeli z pliku rekordowego
 for TT:= 0 to StringGrid1.RowCount do
 begin

 if (FilePos(PR) < FileSize(PR)) then
 begin
 {
 Odczyt pliku będzie przebiegać aż do
 ostatniego rekordu. W innym przypadku
 warunek nie będzie wykonany.

 FileSize(Plik) - Zwraca liczbę zapisanych
 rekordów w pliku.
 }
 end

 // Odczytanie rekordu od 1 do końca
 Seek(PR, FilePos(PR));
```



```
Read(PR, R1); // Odczytanie danych

// Przypisanie poszczególnym kolumnom danych
StringGrid1.Cells[0, 1+TT]:= Trim(R1.Imie);
StringGrid1.Cells[1, 1+TT]:= Trim(R1.Nazwisko);
StringGrid1.Cells[2, 1+TT]:= Trim(R1.Miasto);

end;

end;
CloseFile(PR); // Zamknięcie pliku

end;
end;
```

➤ Kliknij dwukrotnie na klawisz z napisem **Zapisz tabelę** i w wygenerowanej procedurze **OnClick** wpisz kod:

```
procedure TForm1.Button4Click(Sender: TObject);
// Deklaracja zmiennych
var
 R1: PlikRek; // Typ rekordowy
 TT: Longint;
begin
 // Zapisanie tabeli do pliku rekordowego

 DeleteFile(prNazwaPliku); // Usunięcie pliku

 // Powiązanie nazwy pliku z fizycznym plikiem na dysku
 AssignFile(PR, prNazwaPliku);

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

 // Pobranie danych z tabeli i zapisanie do pliku
 for TT:= 0 to StringGrid1.ColCount do
 begin

 // Przypisanie zmiennym "Imie",
 // "Nazwisko", "Miasto" danych z tabeli.
 R1.Imie:= Trim(StringGrid1.Cells[0, 1+TT]);
 R1.Nazwisko:= Trim(StringGrid1.Cells[1, 1+TT]);
 R1.Miasto:= Trim(StringGrid1.Cells[2, 1+TT]);

 // Zapisanie rekordu na koniec pliku
 Seek(PR, FilePos(PR));
 {
 Seek(Plik, Pozycja) - Ustawienie pliku na
 określonej pozycji, na
 której będą dokonywane
```

*operacje odczytu i zapisu.  
 Parametr 'Pozycja' umożliwia  
 podanie numeru zapisanego rekordu.*

```
FilePos(Plik) - Zwraca numer bieżącej pozycji
 zapisanego rekordu.
}

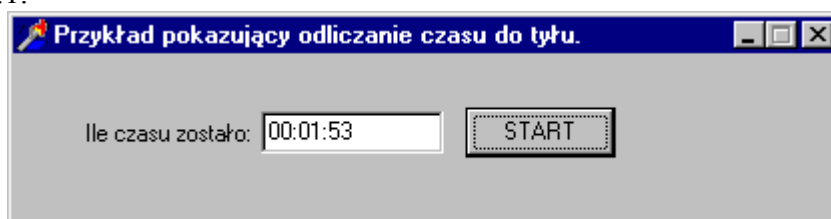
Write(PR, R1); // Zapisanie danych
end;

CloseFile(PR); // Zamknięcie pliku
end;
```

### Ćwiczenie 5.57. Odliczanie czasu w tył





Napisz program, który będzie odliczał czas w tył od zadanej godziny.

Ryśunek 5.57.1.



Przykład znajduje się w katalogu Delphi\Inne\odlicz\_czas.

#### Sposób wykonania:

- Wybierz komponent **Label**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponenty **Edit**  (karta **Standard**);
- Wybierz komponent **Button**  (karta **Standard**);
- Umieść te komponenty na formatce i opisz zgodnie z rysunkiem 5.57.1;
- Wybierz komponent **Timer**  (karta **System**) i umieść go na formatce;
- Napisz funkcje odpowiedzialne za odliczanie czasu w tył (powyżej zdarzenia, np. **OnCreate** = procedurze **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

 // Wylaczenie 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

 // Wlaczanie komponentu TIMER
 Timer1.Enabled:= TRUE;
end;
```

- Kliknij dwukrotnie na komponentcie **Timer** i w wygenerowanym zdarzeniu **OnTimer** wpisz kod:

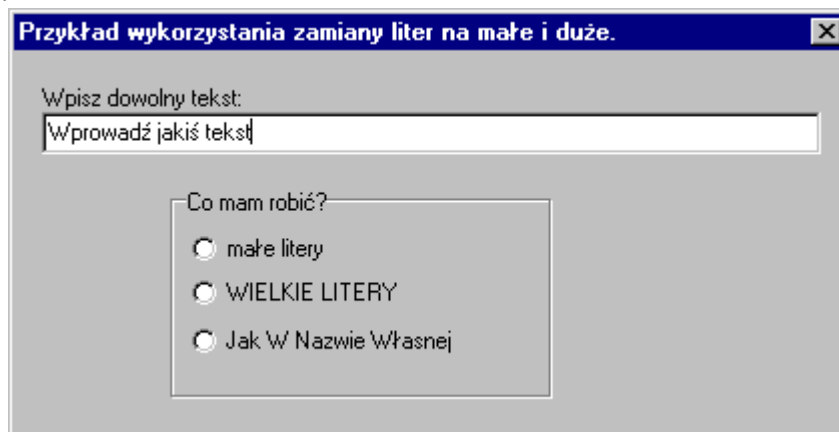
```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
```

```
// Wywołanie funkcji odliczającej czas do tyłu
Edit1.Text:= IleCzasuPozostalo(Edit1.Text);
end;
```

### Ćwiczenie 5.58. 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 5.58.1



Przykład znajduje się w katalogu Delphi\Inne\Litery.

#### Sposób wykonania:

- Wybierz komponent **Label**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponenty **Edit**  (karta **Standard**);
- Wybierz komponent **GroupBox**  (karta **Standard**);
- Wybierz trzy komponenty **RadioButton**  (karta **Standard**);
- Rozmieść i opisz je zgodnie z rysunkiem 5.58.1;
- 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
 tnz. 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:

```

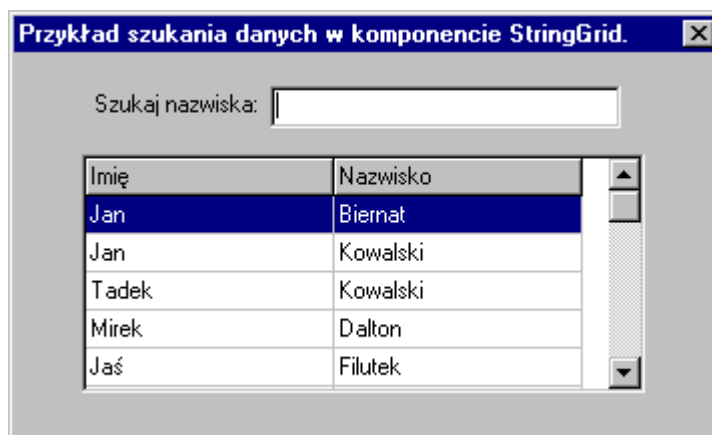
procedure TForm1.RadioButton3Click(Sender: TObject);
begin
 {
 Zmienia tekst na tekst, w którym wszystkie
 wyrazy rozpoczynają się z dużej litery.
 }
 Edit1.Text:= ZmianaZnakow(Edit1.Text, '-', 3);
end;

```

### Ćwiczenie 5.59. Szukanie danych w komponencie StringGrid




Napisz program, który będzie umożliwiał przeszukanie rekordu z nazwiskami.

Rysunek 5.59.1



Przykład znajduje się w katalogu Delphi\Cwicz\Strgrid\_find.

#### Sposób wykonania:

- Wybierz komponent **Label**  (sposób wybrania i umieszczenia komponentu na formatce opisany został w podrozdziale 1.1) z palety komponentów (karta **Standard**);
- Wybierz komponenty **Edit**  (karta **Standard**);
- Wybierz komponent **StringGrid**  (karta **Additional**);
- Rozmieść i opisz je zgodnie z rysunkiem 5.59.1;
- Napisz funkcję odpowiedzialną za wyszukanie danych (powyżej zdarzenia, np. **OnCreate** = procedurze **FormCreate**, a poniżej słowa **implementation**):

```

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 znalezionej komórki
 StringGrid1.CellRect(StringGrid1.Col, StringGrid1.Row);

 // Funkcja zwraca numer wiersza, w który został wskazany
 StringGridSearch:= StringGrid1.Row;

 Break; // Przerwanie i opuszczenie pętli
end;

end;
end;

```

W kolejnym kroku umieść deklarację tej funkcji (metody) w typie obiektowym.

```

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:

```

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;
```

- Zaznacz komponent **Edit** i przejdź do okna **Object Inspektor** (Inspektor obiektów);
- Wybierz zakładkę **Events** (Zdarzenia);
- Kliknij dwukrotnie obok zdarzenia **OnKeyDown**;
- W wygenerowanej procedurze wpisz kod, który umożliwi przejście do komponentu **StringGrid**, za pomocą klawiszy ze strzałkami w dół i górę;

```
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;
```

- Zaznacz komponent **StringGrid** i przejdź do okna **Object Inspektor** (Inspektor obiektów);
- Wybierz zakładkę **Events** (Zdarzenia);
- Kliknij dwukrotnie obok zdarzenia **OnKeyPress**;
- W wygenerowanej procedurze wpisz kod, który umożliwi przejście do komponentu **Edit**, za pomocą dowolnego klawisza:

```
procedure TForm1.StringGrid1KeyPress(Sender: TObject; var Key: Char);
begin
 // Przejście do komponentu Edit
 Edit1.SetFocus;
end;
```

- 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;
```

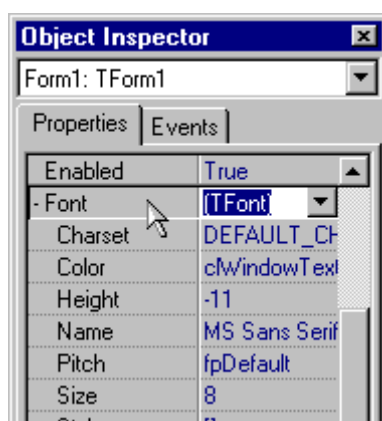
### Ćwiczenie 5.60. Polskie znaki(Ogonki)

*Ustaw stronę kodową polskich znaków dla formatki i sterowników baz danych Paradox.*

#### **Sposób wykonania:**

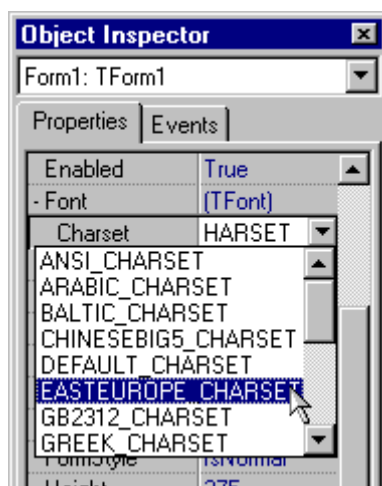
- Przejdź do okna **Object Inspector** (Inspektor Obiektów);
- Rozwiń właściwość **Font** dwukrotnym kliknięciem na tej nazwie – rysunek 5.60.1;

Rysunek 5.60.1



- Rozwiń listę właściwości **Charset**;
- Z listy tej wybierz stronę kodową **EASTEUROPE\_CHARSET** – rysunek 5.60.2;

Rysunek 5.60.2

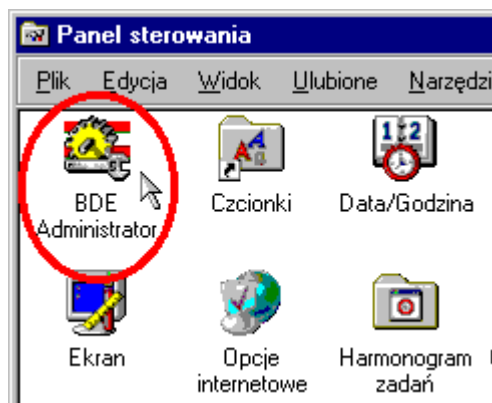


Czcionki ustawione są na stronie kodowej Europy Środkowo-Wschodniej.

W celu ustawienia strony kodowej dla sterowników baz danych typu Paradox należy wykonać następujące kroki:

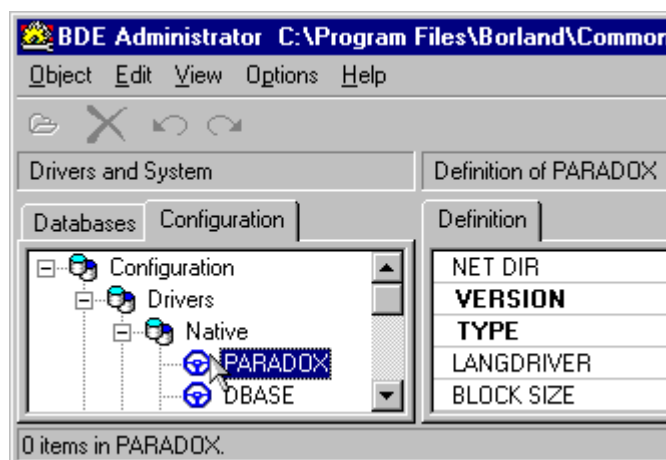
- Wybierz menu **Start/Ustawienia/Panel sterowania**;
- W oknie **Panelu sterowania** wybierz **BDE Administrator** – rysunek 5.60.3;

Rysunek 5.60.3



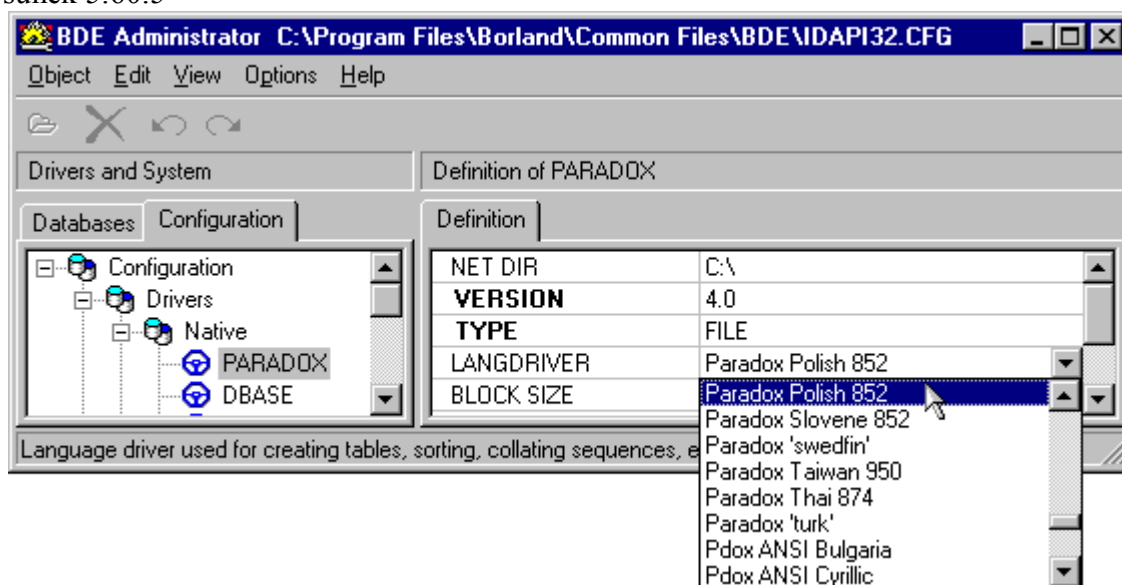
- Wybierz zakładkę **Configuration** (Konfiguracja);
- Rozwiń drzewo **Configuration (Configuration/Drivers/Native/Paradox)**, aż dojdiesz do sterownika **Paradox** – rysunek 5.60.4;

Rysunek 5.60.4



- 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 5.60.5;

Rysunek 5.60.5



Właściwość **LANGDRIVER** sterownika baz danych typu **PARADOX** wykorzystuje polską stronę kodową.

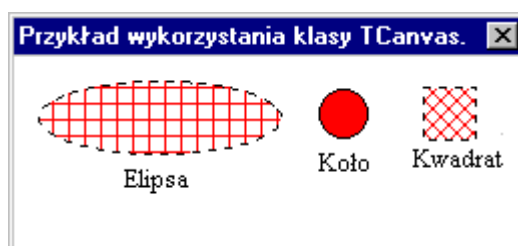
## 6. Ćwiczenia do samodzielnego wykonania

### Ćwiczenie 1:

Do ćwiczenia 5.17 dopisać procedurę wczytującą tabelę do komponentu StringGrid, która będzie wykrywała ilość kolumn i dodawała wiersze.

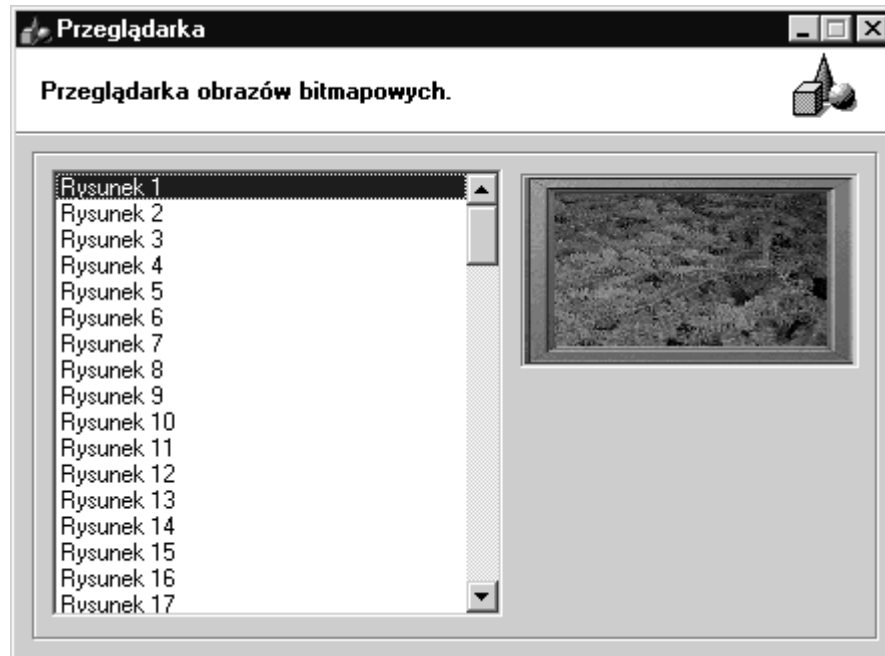
### Ćwiczenie 2:

Napisać program, który losowo będzie rysował figury na formie i podpisywał narysowane figury. Rysunek przedstawia przykładowy program:



### Ćwiczenie 3:

Napisać przeglądarkę obrazków bitmapowych. Rysunek przedstawia wygląd przykładowej przeglądarki.



### Ćwiczenie 4:

Napisz program, który będzie losował liczby i konwertował je na słowa (patrz ćwiczenie 5.37).

**Ćwiczenie 5:**

Napisz program korzystając z programu napisanego w ćwiczeniu 3 dodając pokaz slajdów.

**Ćwiczenie 6:**

Napisz program do nauki tabliczki mnożenia.

**Ćwiczenie 7:**

Napisz program do przechowywania informacji dotyczących: - nazwiska; - imienia; - adresu; - telefonu. Pisząc ten program korzystaj z informacji zawartych w ćwiczeniu 5.38.

**Ćwiczenie 8:**

Napisz program, który będzie wyświetlał tablicę kodów ASCII.

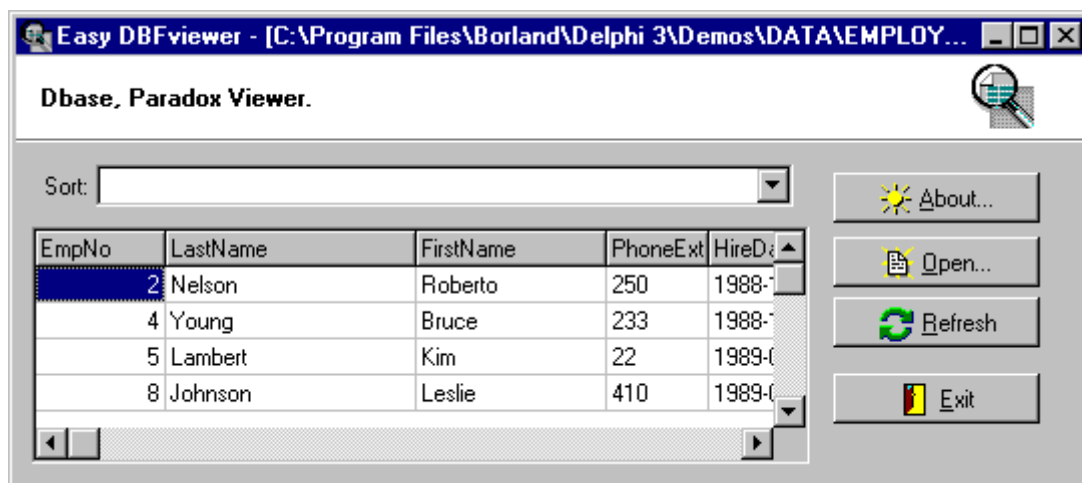
**Ćwiczenie 9:**

Napisz program, który poda nazwę dnia na podstawie daty.

**Ćwiczenie 10:**

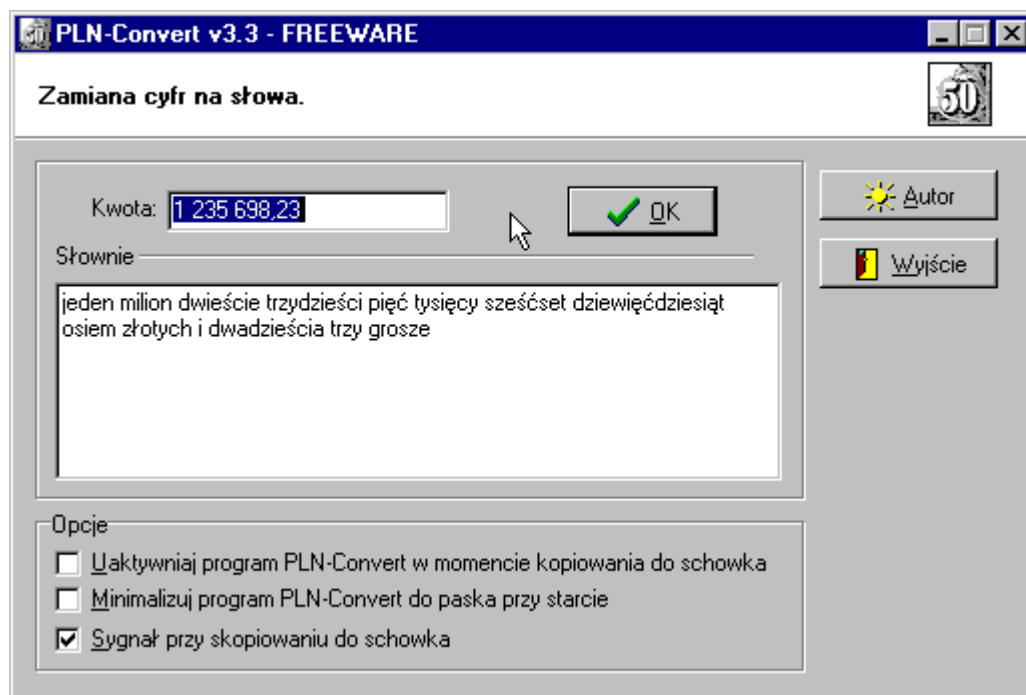
Napisz przeglądarkę do plików bazodanowych Dbase i Paradox'a z możliwością wczytania pliku przez podanie go jako parametr przy wywoływaniu tego programu. Korzystaj z wiadomości zawartych w ćwiczeniu 5.38 i 5.32.

Rysunek przedstawia przykładowy program:







**Ćwiczenie 14:**

Napisać program, który będzie posiadał bazę uczniów wraz z ocenami oraz będzie posiadał możliwość wydrukowania danych.

**Ćwiczenie 15:**

Do ćwiczenia 5.53 dodać procentową prezentację liczb, zapisywanie oraz kopiowanie wykresu do schowka.

**Ćwiczenie 16:**

Napisz program, który będzie porównywał zdobyte oceny uczniów i wyniki te przedstawi na wykresie.

**Ćwiczenie 17:**

Napisz program, który będzie podawał ilość dni pomiędzy dwoma dowolnymi datami.

**Ćwiczenie 18:**

Napisz program, który będzie odtwarzał pliki dźwiękowe pobrane z listy przez jakiś określony czas.

**Ćwiczenie 19:**

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 tego dnia.

**Ćwiczenie 20:**

Napisz program, który będzie liczył średnią z podanych ocen oraz będzie podawał ilość ocen pozytywnych i negatywnych.

**Ćwiczenie 21:**

Napisz tekstową bazę danych, która będzie zawierała następujące pola: imię, nazwisko, adres i telefon oraz będzie umożliwiała sortowanie według tych pól. Program ten należy pisać w oparciu o komponent **StringGrid**.

**Ćwiczenie 22:**

Napisz program, który będzie automatycznie numerował nazwy plików przy zapisywaniu.

**Ćwiczenie 23:**

Napisz program do obliczania równania kwadratowego i  $X_1$ ,  $X_2$ .

**Ćwiczenie 24:**

Napisz program, który będzie automatycznie zmieniał imiona i nazwiska tak, aby zaczynały się z dużej litery, natomiast tytuły naukowe pisał z małej litery.

**Ćwiczenie 25:**

Napisz bazę danych w oparciu o komponent **StringGrid**, która będzie zawierała następujące kolumny: Imię, Nazwisko i Adres. Baza ma umożliwiać dodawanie, edycję, usuwanie, odczytywanie i zapisywanie oraz wyszukiwanie i sortowanie danych.