Kurs języka Object/Delphi Pascal

na bazie implementacji Free Pascal.

AUTOR ŁUKASZ STAFINIAK

Email: lukstafi@gmail.com, lukstafi@ii.uni.wroc.pl
Web: www.ii.uni.wroc.pl/~lukstafi

Jeśli zauważysz błędy na slajdach, proszę daj znać!

Wykład 7: Lazarus GUI

Kompilowanie. Przegląd komponentów.

GUI w *Lazarus Component Library*

- Możemy zbudować GUI z komponentów *Lazarus Component Library* (LCL) bez pomocy *Lazarus IDE*.
- LCL dostarcza obiektu `Application`, w programie głównym go inicjalizujemy, tworzymy okno, i uruchamiamy obsługę zdarzeń (*event loop*).

```
program hello_no_ide0;                    Interfaces dostarcza implementacji GUI.
uses Interfaces, Forms;                   Forms dostarcza TForm
var Form : TForm;                         i zmiennej Application.
begin
   Application.Initialize;
   Application.CreateForm(TForm, Form);   Tworzy okno główne.
   Form.Caption := 'Hello World 0';       Tytuł okna.
   Application.Run;                       Uruchamia pętlę obsługi zdarzeń.
end.
```

- Żeby skompilować przez `fpc`, musimy ręcznie podać ścieżki do LCL, np.:

```
fpc -Fu/usr/lib/lazarus/0.9.30.4/lcl/units/i386-linux
-Fu/usr/lib/lazarus/0.9.30.4/lcl/units/i386-linux/gtk2
./hello_no_ide0.pas
```

- Moduł `Interfaces` dostarcza implementacji interfejsu graficznego przy pomocy konkretnego toolkitu okienkowego jak `gtk2`, `qt` czy `win32`.
- Moduł `Forms` dostarcza klas okien `TForm`, okien/„ramek" `TFrame`, oraz klas i ich instancji dla ekranu `Screen` i aplikacji `Application`.
- Moduł `StdCtrls` dostarcza najczęściej używanych gadżetów jak `TButton`, `TLabel`, `TEdit`, `TCheckBox`, `TGroupBox`, `TListBox`.
- Moduł `ExtCtrls` to użyteczne gadżety jak `TImage`, `TPanel`, `TCheckGroup`, `TTimer`.
- Moduły te używają wielu modułów z bardziej podstawowymi klasami: `Classes`, `Controls`, ...
- Pętla obsługi zdarzeń monitoruje zdarzenia jak np. kliknięcie myszką na komponencie `OnClick` i wywołuje procedurę przypisaną danemu zdarzeniu we właściwości (`property`) danego komponentu.

- Własności komponentów do obsługi zdarzeń mają typ wskaźnika na metodę `procedure(Sender: TObject) of object`.
  - **Wskaźniki na metody** są używane się tak samo jak wskaźniki na procedury (tyle że **wywołają metodę instancji**).
  - <rant> Różnica jest więc techniczna. Podobnie jak domknięcia funkcyjne (*function closures*) z języków funkcyjnych, wskaźniki na metody pozwalają zapakować dane potrzebne wywołaniu, które w przypadku wskaźników na procedury często musiałyby być w zmiennych globalnych. </rant>
- Zazwyczaj procedury obsługi zdarzeń są implementowane jako metody okna na którym leży dany komponent (tzn. klasy pochodnej z `TForm`/`TFrame`).

```
program hello_no_ide1;
uses Interfaces, Forms;
type TEvents = class                    Klasa pomocnicza, bo zdarzenia wymagają...
   procedure CloseApp (Sender : TObject);
end;
var
   Form : TForm; Events : TEvents;
   procedure TEvents.CloseApp (Sender : TObject);
   begin Form.Close end;
begin
   Events := TEvents.Create;
   Application.Initialize;
   Application.CreateForm(TForm, Form);
   Form.Caption := 'Hello World 1';
   Form.OnClick := @Events.CloseApp;          ...wskaźników na metody.
   Application.Run;
   Events.Free;
end.
```

```
program hello_no_ide2;
uses Interfaces, Forms;
type TForm1 = class (TForm)
   procedure CloseApp (Sender : TObject);
end;
var
   Form : TForm1;
   procedure TForm1.CloseApp (Sender : TObject);
   begin Close end;
begin
   Application.Initialize;
   Application.CreateForm(TForm, Form);
   Form.Caption := 'Hello World 2';
   Form.OnClick := @Form.CloseApp;
   Application.Run;
end.
```

Metody okien zwykle obsługują zdarzenia.

- LCL ma dwie hierarchie zawierania: hierarchię `Owner` zarządzania pamięcią i hierarchię `Parent` wyświetlania komponentów.
- Właściciela (`Owner`) ustawiamy w konstruktorze komponentu, zwolnienie właściciela zwolni wszystkie komponenty które on posiada.
- Zamiast ręcznie wywoływać `Create`, możemy użyć `procedure TApplication.CreateForm(InstanceClass: TComponentClass; out Reference);` podając potrzebną clasę oraz zmienną gdzie zapisać komponent.
  - Okna `TForm` powinno się tworzyć przez `CreateForm` – zawiera ona kod wywoływany specjalnie gdy tworzymy `TForm`.
- Rodzica (`Parent`) ustawiamy przez właściwość `Parent` – przypisanie włoży komponent „graficznie” w rodzica.
  - W odróżnieniu od (starszych wersji?) Delphi, nie ma znaczenia kolejność ustawiania rodziców.

```
program hello_no_ide3;
uses Interfaces, Forms, StdCtrls;
type TForm1 = class (TForm)
   procedure CloseApp (Sender : TObject);
end;
var Form : TForm1;
   Button : TButton;
   procedure TForm1.CloseApp (Sender : TObject);
   begin Close end;
begin
   Application.Initialize;
   Application.CreateForm(TForm1, Form);
   Form.Caption := 'Hello World 3';
   Button := TButton.Create(Form);
   Button.Caption := 'Hello!';
   Button.Parent := Form;
   Button.OnClick := @Form.CloseApp;
   Application.Run;
end.
```

Okno jest zarówno właścicielem jak i rodzicem przycisku.

```
program hello_no_ide4;
uses Interfaces, Forms, StdCtrls;
type TForm1 = class (TForm)
   procedure CloseApp (Sender : TObject);
end;
var Form : TForm1;
   Button : TButton;
   procedure TForm1.CloseApp (Sender : TObject);
   begin Close end;
begin
   Application.Initialize;
   Application.CreateForm(TForm1, Form);
   Application.CreateForm(TButton, Button);
   Form.Caption := 'Hello World 4';
   Button.Caption := 'Hello!';
   Button.Parent := Form;
   Button.OnClick := @Form.CloseApp;
   Application.Run;
end.
```

Właścicielem przycisku jest aplikacja natomiast rodzicem okno.

Lazarus IDE

- Lazarus to wygodne IDE w którym możemy
  - ręcznie zaprojektować wygląd aplikacji,
  - wygodnie przeglądać właściwości (properties) komponentów i obsługiwane zdarzenia,
  - wybierać komponenty klikając na sugestywne ikony,
  - edytować źródła, często wskakując od razu do implementacji automatycznie zadeklarowanej metody,
  - korzystać ze wsparcia edytora: podświetlanie składni,
    - indentacja po naciśnięciu Enter (również wstawi `end` po `begin`),
    - autouzupełnianie, w tym intellisense (wybór metody dla obiektu),
    - refaktoryzacja,
  - kompilować, uruchamiać i debugować projekty.

- Klawisz `f12` wywołuje aktualnie konstruowane okno (np. `Form1`).
- Wybieramy komponent z panelu powyżej i klikając na oknie (np. `Form1`) wstawiamy go w odpowiednim komponencie.
- Aktywny (np. kliknięty) komponent ma wyświetlone właściwości / zdarzenia w oknie *Inspektor obiektów* – możemy tam też wybrać aktywny komponent z hierarchii zawierania.
- Dwuklik na wstawionym komponencie wskoczy do implementacji zdarzenia `OnClick`.
- Klikając na ikonce [...] na prawo od wybranego zdarzenia w inspektorze obiektów przechodzimy do implementowania jego obsługi – możemy też wybrać procedurę obsługującą z listy.
- Warto zapoznać się z możliwościami Lazarus IDE i podstawowymi komponentami czytając http://wiki.lazarus.freepascal.org/Lazarus_Tutorial.
- Lazarus tworzy pliki: projektu `.lpi` (XML), programu `.lpr`, osobny moduł dla każdego okna i ramki programu `.pas`, dane komponentów (m.in. rozmieszczenie) na danym oknie `.lfm` (czytelny format).

Plik `.lpr`.

Te ustawienia są też w moim `~/fpc.cfg`,
więc nie włączałem ich do źródeł programów.

```
program hello_with_ide;
{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Interfaces, // this includes the LCL widgetset
  Forms, hello_window
  { you can add units after this };

{$R *.res}

begin
  RequireDerivedFormResource := True;
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Plik .pas.

```
unit hello_window;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls;
type
  { TForm1 }
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.lfm}
{ TForm1 }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Close;
end;
end.
```

Plik `.lfm`.

```
object Form1: TForm1
  Left = 304
  Height = 240
  Top = 142
  Width = 320
  Caption = 'Hello World'
  ClientHeight = 240
  ClientWidth = 320
  LCLVersion = '0.9.31'
  object Button1: TButton
    Left = 42
    Height = 25
    Top = 33
    Width = 75
    Caption = 'Hello!'
    OnClick = Button1Click
    TabOrder = 0
  end
end
```

- Lazarus ma również IDE z linii poleceń w formie polecenia `lazbuild`. Przekazujemy mu nazwę pliku `.lpi` i ono kompiluje cały projekt. Dzięki `lazbuild` nie martwimy się m.in. o ścieżkę do LCL, ale potrzebujemy przynajmniej trywialny plik `.lpi` – rzeczy w nim nie ustawione będą mieć wartości domyślne.

```
<?xml version="1.0"?>
<CONFIG>
 <ProjectOptions>
   <Units Count="1">
     <Unit0><Filename Value="hello_lazbuild.pas"/></Unit0>
   </Units>
   <RequiredPackages Count="1">
     <Item1><PackageName Value="LCL"/></Item1>
   </RequiredPackages>
 </ProjectOptions>
</CONFIG>
```

- Program może wtedy wyglądać tak samo jak przy ręcznym wywoływaniu `fpc`.

```
program hello_lazbuild;
uses Interfaces, Forms;
var Form : TForm;
begin
   Application.Initialize;
   Application.CreateForm(TForm, Form);
   Form.Caption := 'Hello from lazbuild';
   Application.Run;
end.
```

- Oczywiście `lazbuild` kompiluje też pełne projekty Lazarus IDE.

- Dwuklik na `TForm` w inspektorze obiektów przenosi nas do implementacji `FormCreate` obsługującej zdarzenie `OnCreate`, gdzie inicjalizujemy co trzeba.
  - `OnCreate` jest wywoływane po konstrukcji obiektu.
- W większych projektach często tworzy się własne komponenty graficzne które następnie instaluje się w Lazarus IDE jako równoprawne z „wbudowanymi" komponentami LCL.

Przykłady

- Pole tekstowe `TMemo` z pionowym paskiem przewijania, wczytaniem tekstu z pliku i zapisaniem do pliku.
- Rysowanie po `Canvasie` (fraktala z `ChaosGame`), z modyfikacją parametrów przez `TTrackBar`.
- Algorytmiczne wypełnianie formularza – plansza do gry Saper.