PROGRAMOZÁS
VIZUÁLIS/GRAFIKUS FEJLESZTŐI KÖRNYEZETBEN
(MINIMÁLIS TUDNIVALÓK)

A 4GL LÉNYEGE

Szómagyarázat

Vizuális fejlesztői környezet = 4. generációs nyelvhez (4th generation programming language) kapcsolódó környezet. Helyesebb lenne nem „vizuális”, hanem a lényeget jobban kifejező „grafikus” jelzőt használni, hiszen hagyományos környezet esetén is a program számos vizuális jelzést ad ki, amikor a képernyőn kiírja a kérdést, vagy az eredményt. Itt a vizualitás nem tisztán karakteres, hanem „grafikagazdagabb” voltáról van szó.

Egy mondatban…

Grafikus (operációs rendszer) környezeten keresztüli programfejlesztés.

A grafikus kezelőfelület elemei és „szervei”

Alapelemek:

asztal, ikon, ablak stb.

Beavatkozó szervek:

egér, billentyűzet (forróbillentyű-készlet stb.) stb.

Elvárások:

Minden alkalmazáshoz (legalább) egy ablak tartozik, amelyen keresztül vezérelhető, amelyen keresztül kommunikál a felhasználóval. Az ablakokból több is lehet egyidejűleg a képernyőn, ilyen esetben az aktívval lehet kommunikálni.

Az operációs rendszer gondoskodik az ablakok kezeléséről (takarások, mozgatások stb.)

A működés „filozófiája”

A beavatkozó szervek (vagy más, pl. intervallumóra) okozta állapotváltozás ún. *eseményt* generál a számítógépben. Az alapműködés „események érzékelését” és „a megfelelő eseménykezelő végrehajtását” jelenti.

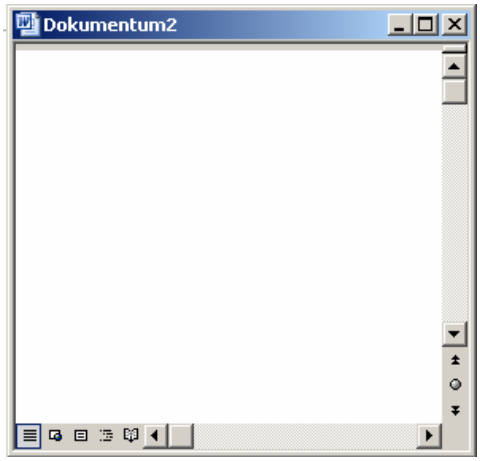
Tehát

1. a „főprogram” feladata: ablak felépítése, kirajzolása (minden ablakkellékkel), és eseményfigyelés, továbbá valamilyen esemény esetén a megfelelő eseménykezelő aktivizálása;
2. az eseménykezelő elvégzi az eseménykor végrehajtandó feladatokat;
3. az operációs rendszer: az –időnként szükségessé váló– ablak újrarajtolást, amit többek közt a kellékek (önállósággal bíró objektumok) „dinamizmusa” is indokol (megszületnek, elenyésznek, előbukkannak, láthatatlanná válnak stb.).

Példák:

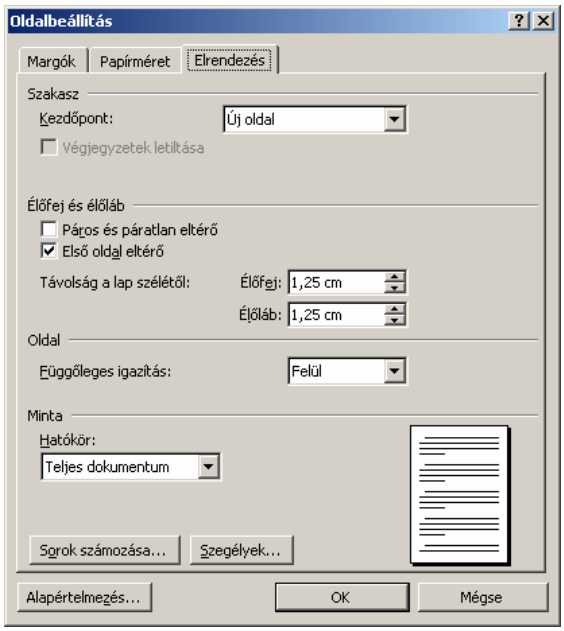
1. Tipikus esemény egy párbeszédablak OK-gombjának lenyomása;
   az eseménykezelő a párbeszédablakban beállított paraméterek tudomásulvétele, felhasználása egy részfeladat megoldásához. Tehát a részfeladat (operációs rendszer feletti) „szokásos” programozói feladat.
2. Tipikus esemény egy ablak címkesorának megragadása az egérrel;
   a hozzátartozó eseménykezelő követi az egeret, miközben az ablakot mozgatja (letörli és újrarajzolja). Ezt a tevékenységet –szerencsére– az operációs rendszer automatikusan végzi.

ABLAKKELLÉKEK ÉS ESEMÉNYEK



1. ábra. Egy „szerény”, tipikus ablak

A baloldali ábrán egy „szerényen dekorált”, de tipikus ablak látszik.
Az ablak alapkellékei:
címkesor; bezáró gomb, minimalizáló gomb, teljes képernyőre kinyitó gomb; gördítő sávok –ha a szükség megkívánja–; keret a méretezéshez.
Mindezen kellékekhez jól ismert használói tevékenységek kapcsolódnak, amelyek az indító események kiváltói.



2. ábra. Egy összetett párbeszédablak

A 2. ábrán egy „kellékekben gazdag” ablak látszik: egy összetett párbeszédablak.

Szómagyarázat:

A *párbeszédablak* segítségével az alkalmazás felszólítja a felhasználót, hogy a továbbfutáshoz szükséges adatokat adjon meg, állítson be. (A adatmegadás sorrendje akár kötetlen is lehet.)

A minta párbeszédablak kellékei:
Füllel kiválasztható oldalak; konstans szövegek; lenyíló listák; jelölő négyzetek, gombok; kép.

GRAFIKUS KÖRNYEZET ÉS OOP

Szómagyarázat

OOP **o**bjektum-**o**rientált **p**rogramozás (object oriented programming)[1]
Objektum = állapot (jellemzők) + operátorhalmaz[2]
Az objektum fenti fogalomkettősére építő, a programműködést folyamatok egymásra hatásaként tekintő programozói szemléletet OOP-nek nevezzük.

A grafikus „kellékek” (ablakok, ikonok, gombok, menük stb.) objektumok. Ui.:

- Rendelkeznek számos nyilvánvaló jellemzővel.
  Pl. ablak, ikon – helye a képernyőn, mérete; gombok – helye az ablakon, mérete, ráírt szöveg …
- Tartoznak hozzájuk események által kiváltott, feladatfüggetlen és –függő tevékenységek.
  Pl. ablak – bezáró gomb lenyomásakor búcsúzás és az ablaklevétel alaptevékenysége; ikon – rákattintáskor az általa szimbolizált alkalmazás elindítás; gomb – a lenyomásakor egy részfeladatot elvégző eljárás végrehajtása, a gomblenyomás animálása mint alaptevékenység…

Az objektumok jellemzőinek és operátorainak „egyesített, általános leírása” az *osztály* (class).[3]

AZ „ÚJSZERŰ” PROGRAMOZÁSI STÍLUSRÓL

Miben tér el a „hagyományos” programozástól a grafikus elemeken alapuló alkalmazás programozása?

Abban, hogy

1. *meg* kell *tervezni* az alkalmazás *kommunikációs felületét* (ablakok, ablakösszetevők), azaz a GUI[4]-t (osztály + „objektum-topológia”: mi, hol, mekkora, milyen … legyen);
2. a kezeléssel összefüggő (várt) *eseményeket* (mi, mikor aktivizálódjon), és
3. *kivitelezni* a megfelelő *eseménykezelők működését* (mi, hogyan működjön).

PROGRAMFELÉPÍTÉS

A szükséges (leíró) fájlok:

- Maga a *főprogram* (forrás és kód).
- Az egyes ablakkellékek *osztály*át (leírását és eseménykezelőit) tartalmazó programegységek: unitok (forrás és kód).
- További, a hagyományos programozás is megkívánta, legfőképpen *típusok*at (pl. listát, sort stb.) megvalósító programegységek: unitok (forrás és kód).

Az persze elképzelhető, hogy egy grafikus objektum nem kíván mást, mint amit „eleve tud”, akkor a hozzátartozó eseménykezelők „üresek”.

---

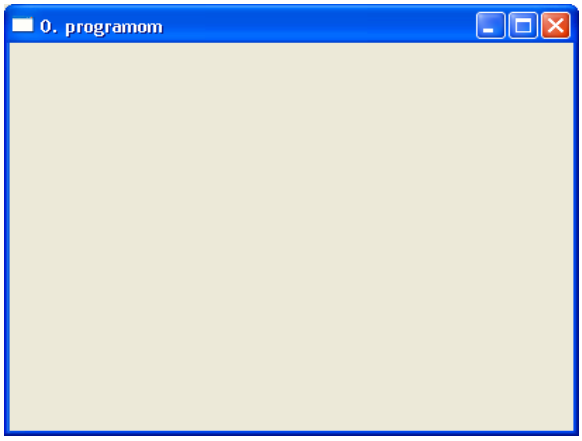
[1] Más szóval: tárgyszemléletű/-középpontú programozás
[2] Más szóhasználat szerint: üzenethalmaz; állapottranszformáció halmaz
[3] Eddig ugyanezt *típus*nak neveztük.
[4] **G**raphic **U**ser **I**nterface

További információkat is meg kell kapjon a fordító program. Legfontosabbak ezek közül:

- *Ablakleíró* (minden egyeshez, külön-külön): mik és milyen jellemzőkkel vannak az adott ablakon.
- *Menüleíró* (minden egyes ablakra került menühöz): milyen a menüstruktúra, mely eljárások tartoznak az egyes menüpontokhoz.

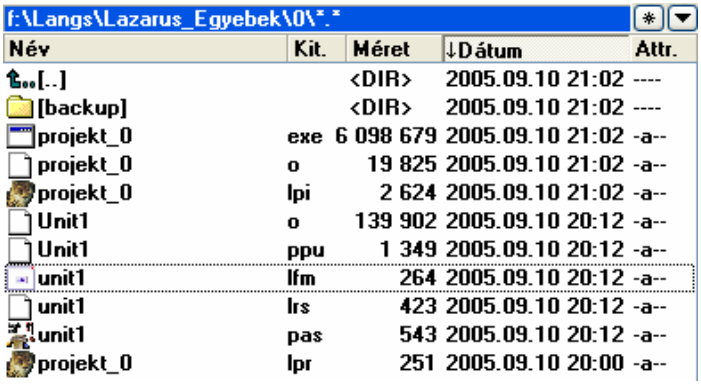
E sok fájlra széttagolt információhalmazt foglaljuk az ún. *projekt*be (projektfájlba). Célszerűen a 4GL fejlesztői rendszere elfedi ennek részleteit, de nem árt valamelyest tisztába lenni tartalmukkal.

1. példa (Windows környezetet feltételezve, a későbbiekben is):



3. ábra. Az alkalmazás futás közben

Egy roppant egyszerű (egy üres ablakból áll) alkalmazáshoz (l. 3. ábra) tartozó projekt, és egyéb fájljai:



projekt_0.exe:
  az alkalmazás maga

projekt_0.lpi:
  a projekt xml leírása

unit1.lfm, ~.lrs:
  az osztály leírása (LazarusForm/-Resource)

unit1.pas:
  az osztály pascal leírása

projekt_0.lpr
  az alkalmazás pascal főprogramja

```
object Form1: TForm1
  Caption = '0. programom'
  ClientHeight = 300
  ClientWidth = 400
  PixelsPerInch = 96
  Position = poDesktopCenter
  HorzScrollBar.Page = 399
  VertScrollBar.Page = 299
  Left = 290
  Height = 300
  Top = 149
  Width = 400
end
```

4. ábra. unit1.lfm

```
{ This is an automatically generated
  lazarus resource file }

LazarusResources.Add('TForm1','FORMDATA',[
  'TPF0'#6'TForm1'#5'Form1'#7'Caption'
  +#6#12'0. programom'#12'ClientHeight'
  +#3','#1#11'ClientWidth'#3#144#1#13
  +'PixelsPerInch'#2'`'#8'Position'
  +#7#15'poDesktopCenter'
  +#18'HorzScrollBar.Page'
  +#3#143#1#18'VertScrollBar.Page'
  +#3'+'#1#4'Left'#3'"'#1#6'Height'
  +#3','#1#3'Top'#3#149#0#5'Width'
  +#3#144#1#0#0
]);
```

5. ábra. unit1.lrs

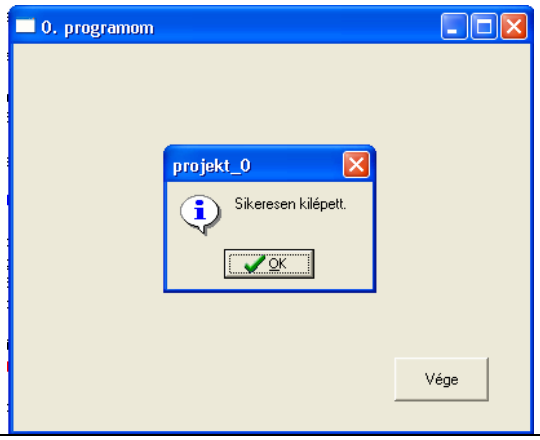
```
{$mode objfpc}{$H+}

uses
  Interfaces, // this includes the LCL widgetset
  Forms
  { add your units here }, Unit1;

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

6. ábra. projekt_0.lpr

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, LResources,
  Forms, Controls, Graphics, Dialogs;

type

  { TForm1 }

  TForm1 = class(TForm)
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{ TForm1 }

initialization
  {$I unit1.lrs}

end.
```

7. ábra. unit1.pas

Egy „tisztességes” kilépést lehetővé tevő gombbal bővítve… és a lényegesebb változások:



8. ábra. A módosított program futása a „Vége” gomb megnyomása után

```
object Form1: TForm1
  Caption = '0. programom'
  …
  Width = 400
  object Vege: TButton
    BorderSpacing.OnChange = nil
    Caption = 'Vége'
    OnClick = VegeClick
    TabOrder = 0
    Left = 295
    Height = 34
    Hint = 'Kilépés'
    Top = 243
    Width = 73
  end
end
```

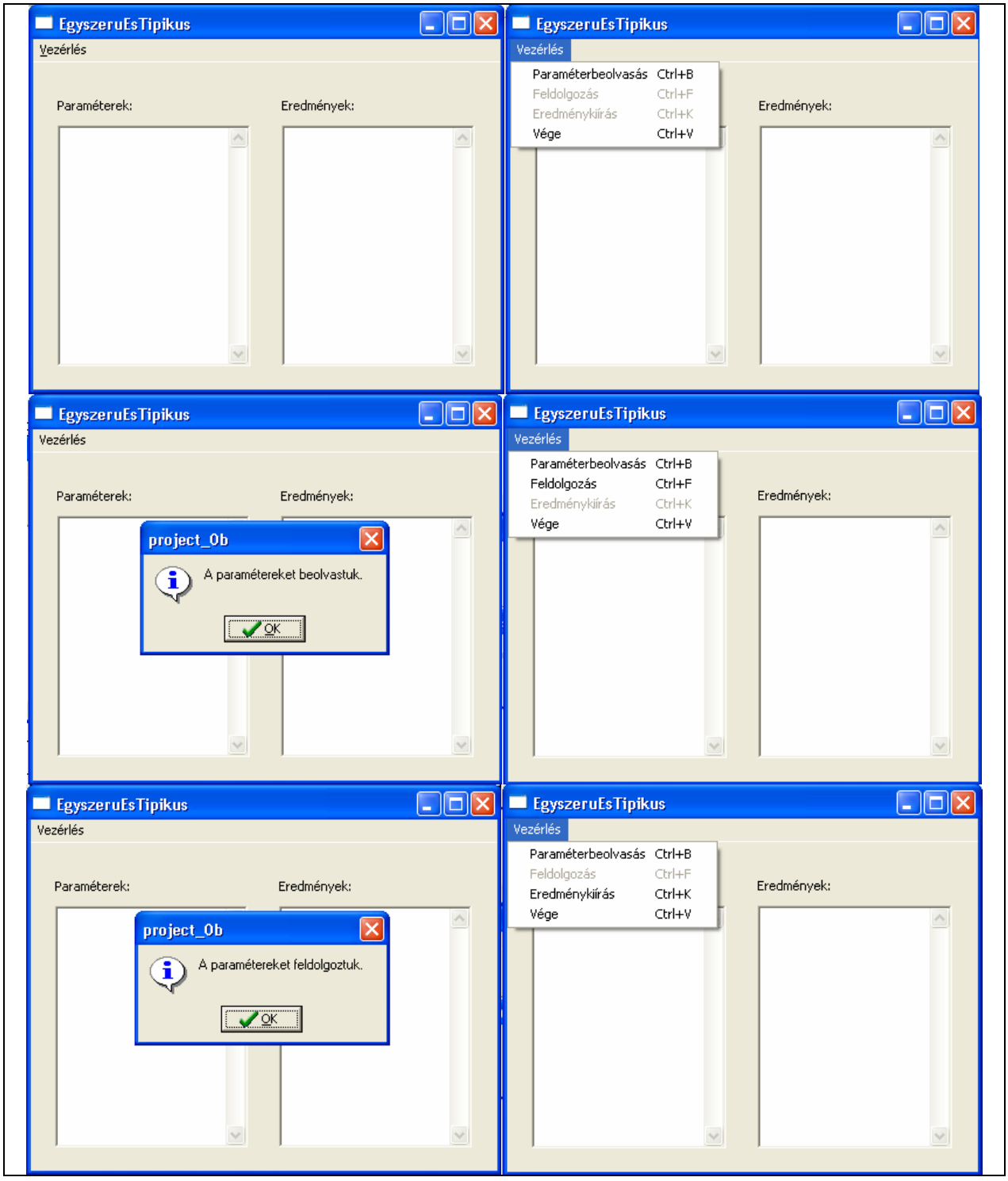
9. ábra. unit1.lfm

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, LResources, Forms,
  Controls, Graphics, Dialogs, Buttons;

type

  { TForm1 }

  TForm1 = class(TForm)
    Vege: TButton;
    procedure VegeClick(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{ TForm1 }

procedure TForm1.VegeClick(
                    Sender: TObject);
begin
  ShowMessage('Sikeresen
                kilépett.');
  close
end;

initialization
  {$I unit1.lrs}

end.
```

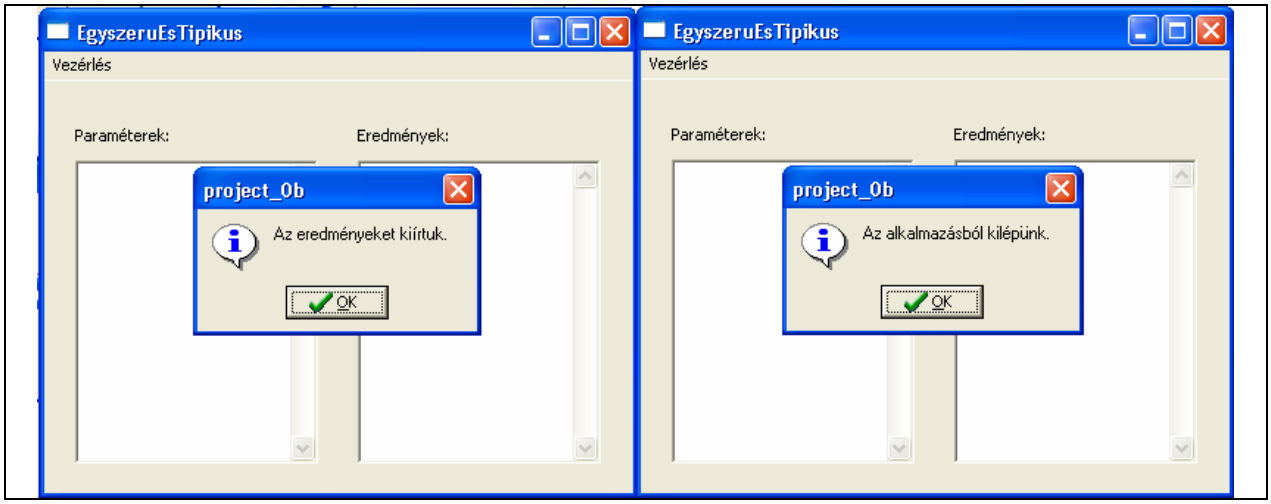
10. ábra. unit1.pas

2. példa:

Egy „klasszikus” alkalmazás[5] tipikus grafikus kellékekkel (menüt és két „memó”-t tartalmazó, de semmi „feladatfüggőt” nem végző, a menüpontok éppen csak jelzik az elvégzendő részfeladatot: beolvasás, feldolgozás, kiírás):



[5] Klasszikus abban az értelemben, hogy a menük csak egy „klasszikus” programoknál megszokott sorrendben választhatók ki.



11. ábra. Néhány jellemző futási állapot

… és fájljai:

```
object EgyszeruEsTipikus:
                        TEgyszeruEsTipikus
  Caption = 'EgyszeruEsTipikus'
  ClientHeight = 280
  ClientWidth = 400
  Menu = MainMenu1
  PixelsPerInch = 96
  Position = poDesktopCenter
  HorzScrollBar.Page = 399
  VertScrollBar.Page = 279
  Left = 284
  Height = 300
  Top = 138
  Width = 400
  object LabelBe: TLabel
    BorderSpacing.OnChange = nil
    Caption = 'Paraméterek:'
    Color = clNone
    Left = 21
    Height = 17
    Top = 30
    Width = 67
  end
  …
```

```
  object MemoBe: TMemo
    BorderSpacing.OnChange = nil
    Lines.Strings = (
      'Ide kerülnek a beolvasott '
      'paraméterek.'
      '1 paraméter = 1 sor.'
    )
    ReadOnly = True
    ScrollBars = ssAutoBoth
    TabOrder = 0
    Left = 21
    Height = 206
    Hint = 'Paraméterértékek'
    Top = 54
    Width = 165
  End
  …
  object MainMenu1: TMainMenu
    object MenuItem1: TMenuItem
      Caption = '&Vezérlés'
      object MenuItem4: TMenuItem
        Caption = 'Paraméter&beolvasás'
        ShortCut = 16450
        OnClick = MenuItem4Click
      end
      …
  end
```

12. ábra. unit1.lfm (részletek)

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, LResources, Forms,
    Controls, Graphics, Dialogs, Menus,
    StdCtrls;

type

  { TEgyszeruEsTipikus }

  TEgyszeruEsTipikus = class(TForm)
    LabelBe: TLabel;
    MemoBe: TMemo;
    LabelKi: TLabel;
    MemoKi: TMemo;
    MainMenu1: TMainMenu;
    MenuItem1: TMenuItem;
     (* Vezérlés -- Főmenü *)
    MenuItem4: TMenuItem;
     (* Paraméterbeolvasás *)
    …
    procedure MenuItem4Click(
                      Sender: TObject);
    …
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  EgyszeruEsTipikus:
                  TEgyszeruEsTipikus;

implementation

{ TEgyszeruEsTipikus }

procedure TEgyszeruEsTipikus.Menu
         Item4Click(Sender: TObject);
  (* Paraméterbeolvasás *)
begin
  (*itt olvassuk be a paramétereket*)
  MenuItem2.Enabled:=True;
    (*a Feldolgozás most már
      kiválasztható*)
  MenuItem6.Enabled:=False;
    (*Paraméterbeovasás után az Ered-
      ménykiírás nem választható ki*)
  ShowMessage('A paramétereket
               beolvastuk.');
end;
…
initialization
  {$I unit1.lrs}

end.
```

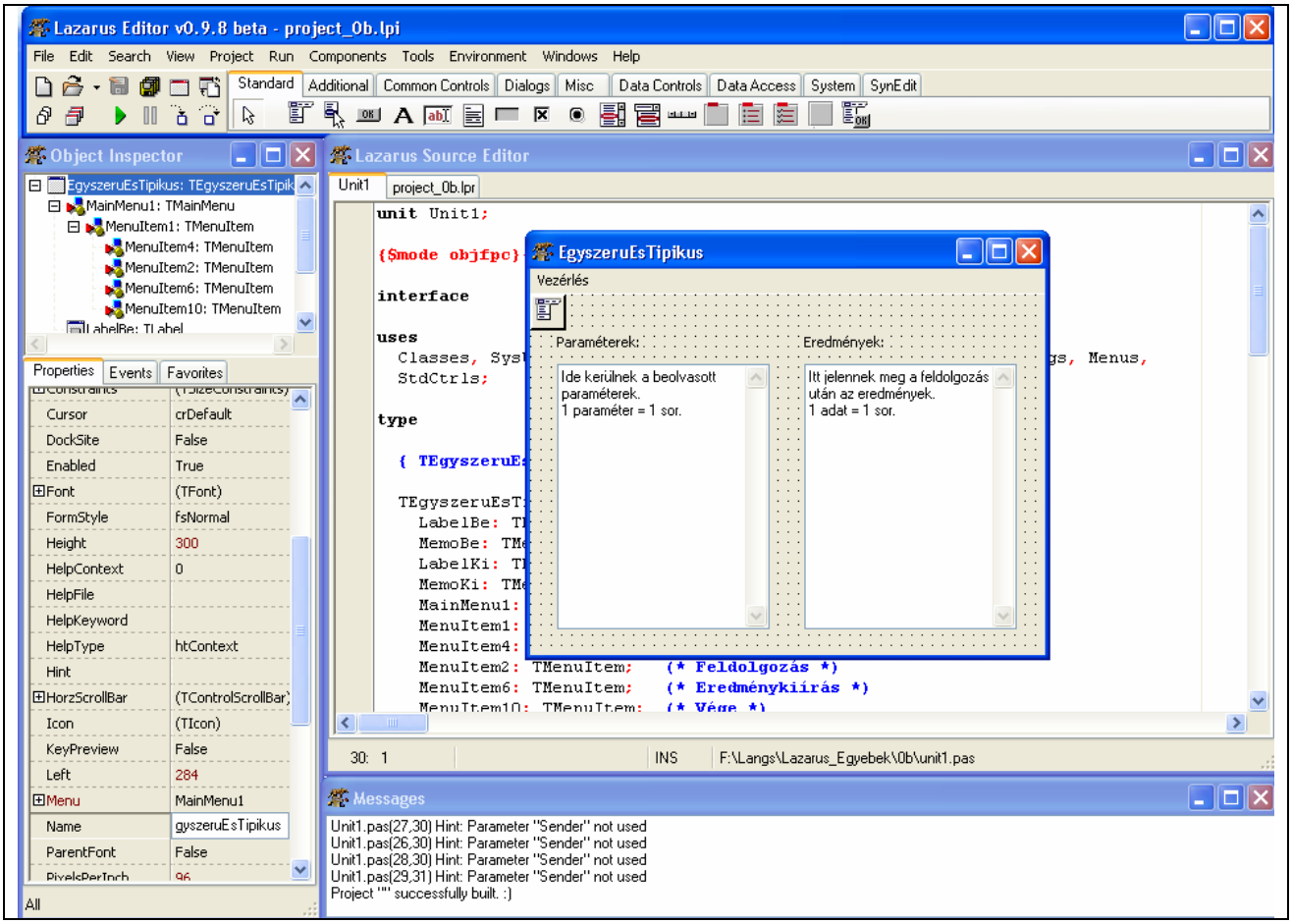
13. ábra. unit1.pas (részlet)
(*A* `{}` *megjegyzések automatikusak, a* `(**)` *fejlesztőiek.*)

GUI

A fejlesztői felületet menetközben mutatja a következő (14. ábra) ábra. Ezen már feltűnnek a fejlesztői környezet leggyakrabban használt ablakai, és egy –a legutóbbi példában bemutatott– alkalmazás.

A legfontosabb kezelőeszközöket tartalmazó ablakok:

- Menü és eszköz-sor (komponens-paletta) – sok-sok ismerős és 4GL-specifikus menüponttal, eszközikonnal
- Object Inspector – az adott objektum attribútumainak beállítása
- Source Editor – az adott osztály eseménykezelőinek pascal kódjának megszerkesztése
- Messages – a fordító üzenetei
- Form – az alkalmazás ablakának „tartója”, amin kell elhelyezni a grafikus (és egyéb) komponenseket



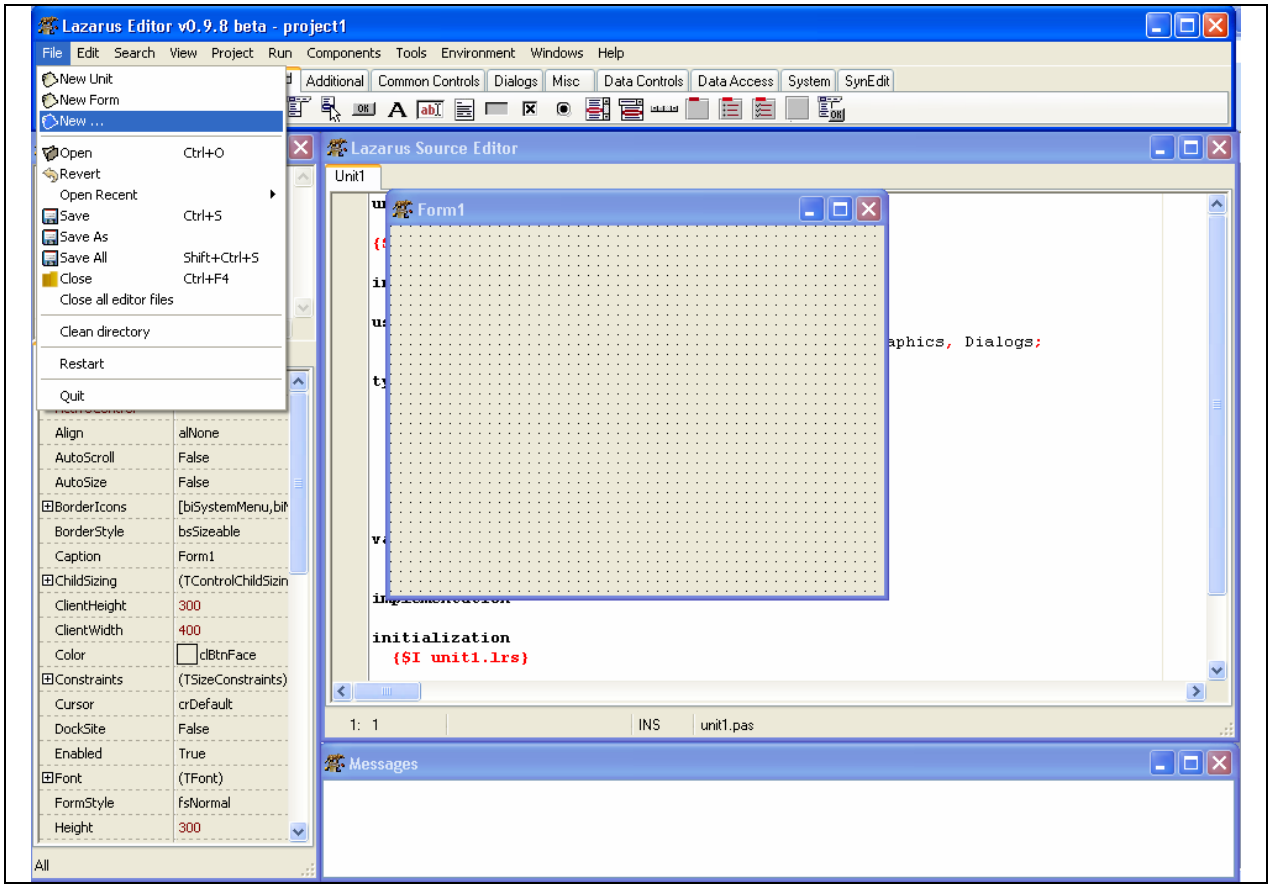
14. ábra. Egy pillanatkép

PRAKTIKA

A kezdő lépések

1. A Lazarus () elindítása (az utána kitárulkozó képet lásd a 15. ábrán)
2. New + Application
3. Az eszközpalettáról kiválasztjuk a kívánt (grafikus) eszközöket,
4. beállítjuk kezdő állapotukat,
5. kiválasztjuk a kívánatos eseményeket, s elkészítjük az eseménykezelő eljárás kódját,
6. kimentjük (azaz Save As a megfelelő alkönyvtárba)
7. lefordítjuk, futtatjuk (Run + Run = F9)

Javaslom, hogy a futtatást is a kód elindításával végezzük (azaz ne használjuk ki a Lazarus futtató rendszerét). Csak ha elkerülhetetlen, akkor nyomkövessünk.



15. ábra. A kezdőkép

A legfontosabb grafikus eszközök osztályai a palettákon

16. ábra. A Standard eszközpaletta

`TMainMenu` – menü

`TLabel` – kísérő szöveg

`TMemo` – hosszú szöveg

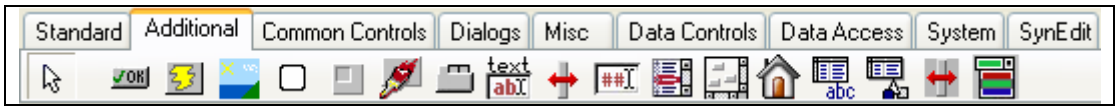
`TListBox` – listából választás

`TButton` – nyomógomb

`TEdit` – egysoros szöveg

`TCheckBox`, **`TCheckGroup`** – választógomb(ok)

`TRadioButton`, **`TRadioGroup`** – rádiógomb(ok)



17. ábra. Az Additional eszközpaletta

`TStringGrid` – szövegek mátrixos elrendezésben

18. ábra. A Common Controls eszközpaletta

`TPageControl` – fülekkel lapozható „füzet”

Pozitív és negatív praktikák –
a Lazarus „nem-szeretemségei, szerencsétlenségei” és egyebek

- ✓ Célszerű kialakítani a saját névkonvenciónkat, amelyből ránézésre kiderül, hogy milyen természetű és milyen sajátos célú objektumról van szó, de nem túl hosszú (pl. `lbBe:TLabel` a felajánlott hosszú és semmitmondó `label1` helyett, `mmBe:TMemo` az automatikus `memo1` helyett stb.).
- ✓ Érdemes a lényeget külön unitba elkülöníteni, amelyek rutinjait a formhoz tartozó eseménykezelők hívnak meg; ekkor persze a saját unitot szerepeltetni kell a form `uses` deklarációjában.
- ✓ …
- ☠ Nem szereti, de nagyon nem, hogy egy kontrollnak utólag változtatják meg a nevét; pl. nem írja át a hozzátartozó eseménykezelő metódusok nevét sem, a tulajdonságokra hivatkozásokat sem; ezen némileg segít a névre kattintva előhívott helyi menüből kiválasztható `Refactoring/Raname`.
- ☠ Futási hiba esetén sokszor úgy tűnik, mintha nem futna, vagy éppen magába mélyedne, ekkor `CTRL+F2`-vel le kell lőni (ha hagyja!).
- ☠ Van úgy, hogy fejlesztett programot lelőttük, mégis mintha futna, ez megakadályozza, hogy újrafordítsuk, pontosabban az exe-t felülírjuk; ez gyakorta akkor fordul elő, amikor a LAZARUS környezetben (`F4`, vagy `RUN`) indított futás közben egy kivételdobás következik be. Ez különösen nyomkövetés közben okoz kellemetlenségeket. Ilyen esetben kézzel sem lehet törölni az exe-t, a ’`project1.exe`’ (az aktuális alkalmazás) és a ’`gdb.exe`’ (a debugger) folyamatot kézzel kell leállítani!
- ☠ A nyomkövető számára az indexelés mást jelent, mint a végrehajtáskor; így a (watch ablakban) megfigyelt adatok értelmezésénél óvatosnak kell lenni.
- ☠ A kivételes események kezelését ne a LAZARUS környezetbe próbáld ki! Ott nem OK, pedig a futó alkalmazás lehet OK.
- ☠ Fordítás közben nem szabad még mozgatni sem az ablakokat (formokat), mivel az állapotuk befolyásolhatja a kódot; e miatt kivétel képződhet a fordítás közben, s elszállás.
- ☠ Debug közben a program formjait nem szabad az asztalra tenni, mert futási hibát okoz, sőt fordításkor sem.
- ☠ Úgy tűnik: a **`not`** és az **`in`** precedenciája nem logikus; hibátlanul lefordul, de rosszul hajtódik végre:

```
"if not parSorsz in [0..maxParSorsz] then
     Raise ENotExistIndex.Create('Ilyen paraméter nincs.');"
```

amíg így helyesen is fut:

```
"if not (parSorsz in [0..maxParSorsz]) then
     Raise ENotExistIndex.Create('Ilyen paraméter nincs.');"
```

- ☠ Az **`in`** operátor paraméterei csak BYTE típusúak lehetnek; „szélesebb” számtípusúakat automatikusan ilyenre konvertál; ebből baj lehet, ha negatív szám volt, mivel pl. -1-ből 255 lesz.

- A StringGrid-beli Cells tartomány indexelése oszlop-sor sorrendben; ez nem hiba, csak szokatlan.
- …