vizuális

1. előadás   IX. 14.

aries. elte. hu / ~ dream → letöltések → delphi

Irodalom:
Marco Cantu : Delphi 5 mesteri szinten

H 11^{10} OOP

Tételek:

1) Betokozás , Kivételek kezelése

2) Polimorfizmus , a Delphi fejlesztő környezete

stb. → LETÖLTHETŐ

Az OOP alapjai:

3 fő ismérv:

1) - Betokozás - bezárás → osztályok valósítják meg (encapsulation)

2) - öröklődés (inheritance)

3) - Többalakúság (polimorfizmus) késői kötés „late binding"-nek is nevezik

1) • eljárásorientált programozásnál voltak lokális és globális változók, eljárások és fgv-ek → nem tükrözik a valós világot

• a külvilágnak nincs sok köze ehhez, ezek működéséhez

• minden osztály rendelkezik valamilyen nagyságú interface-szel, amivel kommunikál

• az objektumosztály belügye a hogyan? (ezt a külvilág nem tudja.)

2) öröklés → a régi dolgok kiegészítődnek, a metódus nem változik, csak a végrehajtás

- az ismétlődő metódusok használatát segíti
- lehetőség van módosítani, változtatni, kiegészíteni a gyermekosztályt ⇒ a specifikumokat lehet öröklöttekre

3) • virtuális metódusok segítségével lehet megvalósítani
- nem dől el fordítási időben, h. 1 metódus hívásakor mit kell használni, mi hajtódjon végre (milyen kód) ⇒ futási idő közben dönt ez ki

A osztály egy lehetséges definíciója:

Egy felhasználó által megadott adattípus, melynek van állapota (megjelenése) és vannak műveletei

Belső adatokkal és metódusokkal rendelkezik.

Több hasonló objektum jellemzőit és viselkedését írja le.

A osztály = típus → ezt példányosítani kell, ezt használjuk majd futási időben.

Az objektum

A osztály egy példánya v. egy osztály által meghatározott változó.

{A objektumok és osztályok kapcsolata a változók és adattípusok kapcsolatához hasonlítható.}

```
type ...
     (osztály)

var                                  ← mutató, ami sehová nem mutat
    objnev : típus (osztály)

begin
    objnev := osztály . konstruktor
end;
:
```

Egy osztály deklarálása Példa

```
Type
    TDate = class
    month, day, year : integer ;
    procedure setvalue (m, d, y: integer) ;
    function ...                          → ezek osztályok és metódusok
        :
```

implementálásnál értéket adunk az év, hónap, nap-nak.

- objektum és objektum példány között van különbség.
- akkor van objektum példány, ha hozzárendeltünk az objektum-hoz memóriát. (ez a create -vel történik)

  pl.: ADay := TDate.create

A.Day.free → megszűnik, mint objektum példány, de az objektum megmarad, amíg tart a hatóköre (függvény v. eljárás tartalmazza, esetleg globális változó-e.)

Hozzáférési lehetőségek:

2. előadás

Hozzáférési lehetőségek

- private (privát) csak ő látja, a gyermekosztályok sem

  ezen mezők és metódusok nem érhetők el az osztályleíró egységen kívülről

- public (publikus)

  a program tetszőleges pontján hozzáférhetők

- protected (védett)

  korlátozott láthatósággal rendelkezik

  a leszármazott osztályok láthatják, mások nem. szigorúbb, mint a public, de lazább, mint a private

ezen belül előbb vannak az adatdefiníciók → szekciók vannak

A SELF kulcsszó

A metódus rendelkezik egy implicit paraméterrel, ami az aktuális objektumra mutat → SELF

pl.: A Tdate osztály metódus használatánál

```
self.month := 1;
self.day := 10;
```

Komponensek dinamikus létrehozása

OWNER: tulajdonos
Parent: szülő

Mindkét esetben az aktuális formot adjuk meg, és ennek a legegyszerűbb módja a Self kulcsszó használata.

elnevezés: futás közben létrehozunk egy kétdimenziós tömböt, amelyben az elemei nyomógombok.

owner: tulajdonos tulajdonság, a konstruktor meghívásakor eldől

parent: meg kell mondani, h. ki a szülő, mert azzal a

keretén fog megjelenni, amit dinamikusan létrehozunk

Minta a létrehozásra

```
Var   btn : Tbutton;          -> változó = objektum
                              -> objektumosztály neve, a rendszer tartalmazza
Begin
   btn := Tbutton.create(self);      -> megadjuk ki az owner
                                     -> lefoglal ennek memóriát = létrehozza
                                        objektumpéldányunk van.
   btn.parent := self;               -> szülő tulajd.
   btn.left := 30;                   -> balról mennyit húzzon be
   btn.top := 20;                    -> fentről
   btn.width := btn.width + 50;
   btn.caption := 'Ez egy gomb';
end;
```

Ha 1442db nyomógombot akarunk létrehozni, => mindegyik nyomógombra meg kell hívni a konstruktort.

Konstruktor:

Egy objektum számára szeretnénk memóriát foglalni, azt a CREATE metódussal tehetjük, mely az objektum KONSTRUKTORA.

A destruktor a FREE metódus lesz. (-> ellenőrzi, hogy létezik-e, mielőtt felszabadítja)

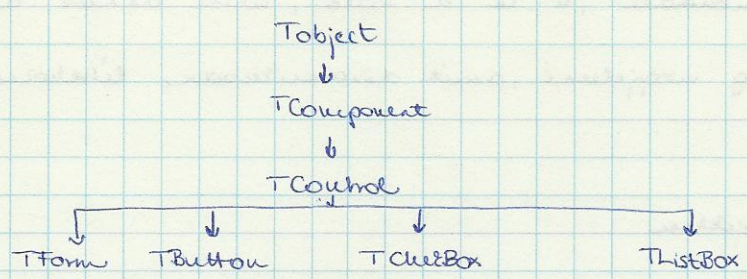
Az alapértelmezett a Tobject.CREATE metódus, minden adatot nullára állít. Ha nem így szeretnénk, akkor saját konstruktort írhatunk.

Öröklődés:

Egy új osztályt közvetlenül egy régi alapján definiálunk.

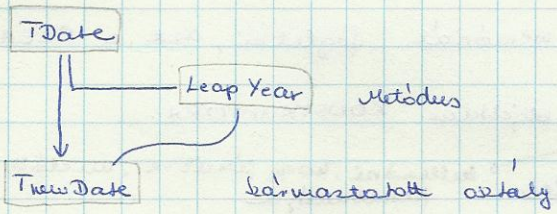
```
Type
Tnewdate = class (TDate)
public
     function GetText: string;
end;
```

Alap öröklődés:



Késői kötés és típuskompatibilitás:

- Ha metódus szerepel az osztály deklarációjában ⇒ egyértelmű, hogy az leszármazási végére.
- Ha nem, ⇒ az öröklési gráfban visszafelé kell keresni, hogy hol történt a deklaráció.




```
var ND : TNewDate
    ND := TNEWDate.Create;
    if ND.LeapYear then ...
```

Létrehozzuk az ND nevű objektumot, és utána meghívjuk a LeapYear metódust → és „körbenéz" és visszatérés az öröklési fában.

Vizuális form öröklődés:

Egy formot származtatunk egy másik formból, majd kiegészítjük az új komponensekkel.

type ⟶ type

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

```
TForm2 = class (TForm)
private
 { private declarations }
public
 { public declarations }
end;
```

↳ TFORM2-t a TForm1-ből öröklekjük.

Létrehozunk egy alapformot, ezt öröklekjük majd tovább.

AB-esetében öröklekjük formra, így egyszerűsödik a munkánk.

A kivételek kezelése:

- Try - except - on error do - end;

```
try
   X := Y/Z ;
except
   on EZeroDivide do
      X := 0 ;
end; //try
```

- ↳ try és except szavak közé írjuk azt /azokat a sort / sorokat, amiben / amikben hiba / hibák keletkezhet(nek).
- except után on és do kulcsszavak közé kell írni a kibarádot (kódokat) és utána, hogy mi történjen. Ha több dolgot akarunk megadni → Begin end közé.
- a zárójeleket end zárja.

- Előnye: ha mi találjuk meg a hibát, akkor magyar nyelvű hibaüzenetet írni, lekezelhetjük a hibát ...stb.
- ha több dolgot kell csinálni a try - except között, és a második sorban hiba van, a hibakódokhoz ugrik, lekezeli, és nem ugrik már vissza a 3., 4. sorra. Ilyenkor minden sort try - except közé kell írni. (Nem lehet goto-t használni, bár a delphi tudja)

Saját kivételek, a kivétel kiváltása

- Type

```
EArrayFull = class(Exception)
If MyArray.Full then
  raise EArrayFull.create('A tömb megtelt');
```

A finally blokk

A finally blokk mindenféleképpen végrehajtódik a try blokk után, még akkor is, ha valamilyen kivétel váltódott ki.

```
Try
    file műveletek
    számítások
    objektumok
                    } Try blokk
Finally
    file lezárása
    memória felszabadítása
                    } Finally blokk
                    → mindenképpen végrehajtandó utasítások
end;
```

- a try - except és a try - finally kötözet egymásba ágyazhassuk
- ha a try-ban hiba keletkezik, a finally mindenképpen végrehajtódik, de ha a finally-ban hiba keletkezik ⇒ problémás.