Alpár Tibor L.

Mérnöki szoftverek készítése

Mérnöki feladatok megoldása programozással Delphi/Kylix rendszerben

2010. szeptember 4.

Szakirodalom

- Angster Erzsébet: Az objektumorientált tervezés és programozás alapjai
- Baga Edit: Delphi másképpen
- Eric Harmon: Delphi/Kylix alapú adatbázis-kezelés
- Marco Cantú: Delphi 7 mesteri szinten I-II

Források

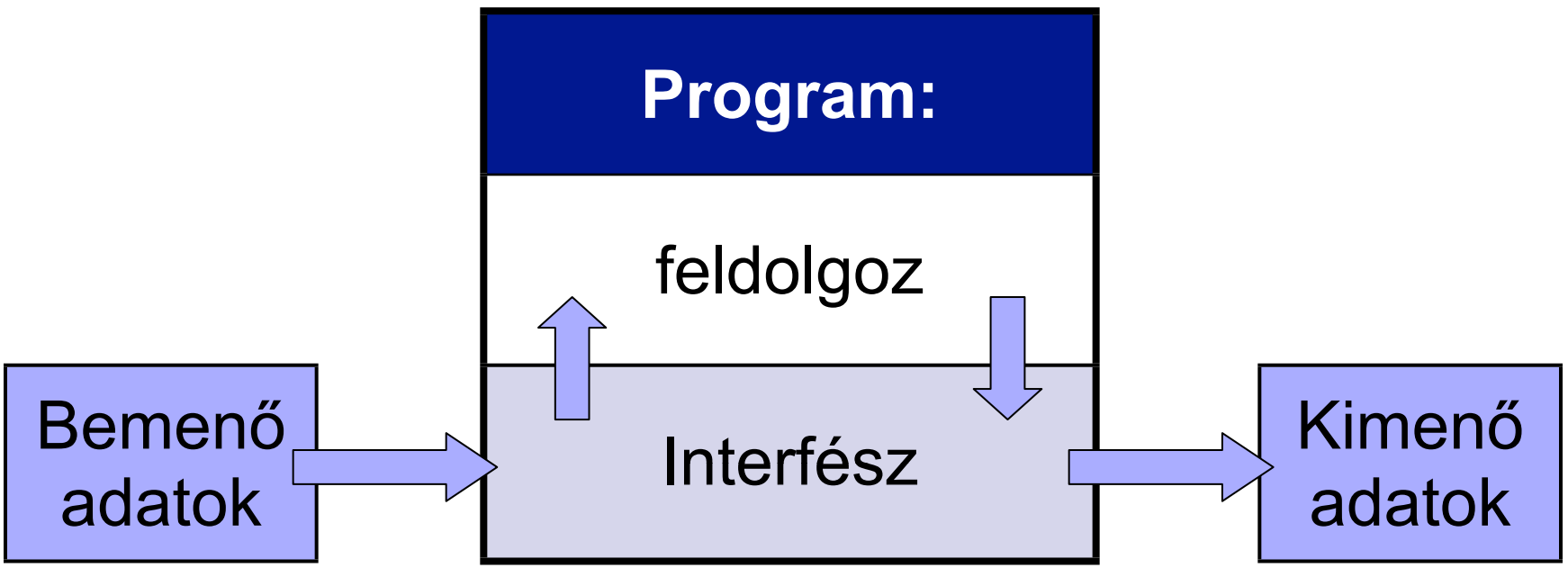
- Torry's Delphi Page: www.torry.net
- Delphi Super Pages: delphi.icm.edu.pl
- Richey's Delphi-Box: inner-smile.com

- Magyar levelező lista: http://list.prog.hu

I. SZOFTVERKÉSZÍTÉS

- nyelvek
- generációk
- szoftverek minősége
- algoritmus

Mit is csinál egy program



Programozási nyelvek - szintek

- ***Alacsony szintű nyelvek***
  - a számítógéphez közel álló nyelvek
    - gépi kód
    - Assembly
  - Előnyük
    - kis méret
    - gyors futás
  - Hátrányuk
    - nehezen átláthatók
    - nehezen érthetők és programozhatók
    - minden egyes számítógépfajtára eltérőek

Programozási nyelvek - szintek

- **A *magas szintű nyelvek***
  - Pascal , Basic, C, C++, Fortran, Java, SmallTalk, stb.
  - Előnyeik
    - közelebb állnak az emberi gondolkozáshoz
    - „nyelvtani” szabályaik többnyire megegyeznek minden géptípusra
  - Hátrányaik
    - nagy fordított futtatható állomány
    - lassabb futási sebesség
    - bizonyos nyelveknél (pl. Visual Basic) külön szubrutinkönyvtárakat kell az EXE mellé vinni

Programozási nyelvek - szintek

- Magas szintű nyelvek működése
  - ***forráskód****ot* (= source code)
  - ***fordító*** (= compiler) fordítja gépi kódra
  - ***tárgykód*** (= object code) keletkezik, amely relatív memóriacímeket tartalmaz
  - ***programszerkesztő*** (= linker) [*más programokkal összeszerkesztve*] teszi futtathatóvá
  - a *linker* és a *compiler* egybeépítve a fordítóprogram

Programozási nyelvek - generációk

- Generációk szerint lehetnek 1G, 2G, 3G és 4G nyelvek.
- A számítástechnika fejlődésével egyre egyszerűbben használhatóvá, sokak által könnyen kezelhetőkké váltak.
- 4G nyelv pl. Informix, Oracle vagy a LabView
  - Ez teljes, előre megírt, grafikusan megjelenített modulokból áll
  - az egyes elemeket csak össze kell kötni, megadni a bemenő adatokat, és megkapjuk a keresett eredményt

[- ez nem valódi programozás -]

A szoftver minősége

- ***Helyesség*:** a szoftver feladat specifikációja helyes, és a szerint működik
- ***Hibatűrés*:** szokatlan esetekben is megfelelően működik
- ***Karbantarthatóság, bővíthetőség*:** legyen könnyen módosítható, javítható, ill. az újabb igényekkel könnyen lehessen kibővíteni
- ***Újrafelhasználhatóság*:** legyen a teljes szoftver vagy annak egyes részei más szoftverekben újrafelhasználható

A szoftver minősége

- ***Kompatibilitás*:** fontos, hogy a szoftverek egymással kommunikálni tudjanak, egymás adatait fel tudják használni
- ***Felhasználóbarát*:** legyen a szoftver egyszerűen, egyértelműen, logikusan használható, szép kivitelű.
- ***Hordozhatóság*:** ha a szoftver könnyen áthelyezhető más szoftver és hardver környezetbe.
- ***Hatékonyság*:** ha jól kihasználja a rendelkezésre álló erőforrásokat, így a lehető leggyorsabban hajtja végre feladatait.
- ***Ellenőrizhetőség*:** ha a tesztelése könnyen

Algoritmus, program

- Azokat az utasításokat, amelyek egy adott feladat megoldásához vezetnek, **algoritmus**nak nevezünk.
- A számítógép által értelmezhető nyelven megírt algoritmust nevezzük **program**nak.

Algoritmus - elvárások

- véges számú lépésből áll,
- minden lépés legyen egyértelműen végrehajtható,
- hivatkozhatunk összetett lépésekre is,
- a végrehajtandó utasításoknak legyen célja,
- általában vannak bemenő adatai, és kimenő adatai.

II. Objektumorientált programozás

- Eseménykezelés és üzenetvezérlés a Windows-ban
- Adattípusok az ObjectPascal-ban
- Az objektumorientált programozás alapjai
- Objektumok, osztályok
- Öröklődés

II.1. Eseménykezelés és üzenetvezérlés - fogalmak

- A modern operációsrendszerek objektumorientált elven működnek.
- Minden ablakokban történik, amely ablakok objektumoknak tekintendők, mivel:
  - vannak adataik (pl. nevük, színük, méretük, gombjaik, beviteli mezőik,…), és
  - van viselkedésük (külső és belső eseményekre reagálnak – egérkattintás, túlcsordulás, …).

Fogalmak

- **Multitasking (= többfeladatúság)**
  több alkalmazás is futhat párhuzamosan, amelyek lehetnek különbözőek, de akár egy alkalmazás több példánya is. A 32 bites Windows ún. *preemptive multitasking*-ot használ, ami azt jelenti, hogy minden feladat (= *task* → értsd: alkalmazás) csak egy adott időszelet idejére kapja meg a rendszer erőforrásait.
- **Multithreading (= több szálon futás)**
  A 32 bites alkalmazások esetén egyszerre több programrész is futhat. A többfeladatos (multitasking) működésben futó kódként résztvevő programrészt *thread*-nek (= szál) nevezzük. Így egy adott programon belül egyszerre több szál is működhet.

Fogalmak

- **Esemény** (= *event*)
  Olyan történés, amelynek hatására megváltozhat egy objektum állapota. Pl.: jel, hívás, őrfeltétel, idő. Az objektumorientált programozás alapja, hogy az objektumok vezérlése események hatására történik. Ezt hívjuk **eseményvezérelt** (*event driven*) működésnek.
- **Üzenet** (= *message*)
  Az üzenet egy, a Windows által generált rekord, amelynek az adatai az őt kiváltó esemény típusára, paramétereire vonatkoznak. Az üzenet lehet:
  - külső (egér kattintás, billentyű leütés, …),
  - belső (időzítő = timer, ablak átméretezés = resize, …).

Fogalmak

- Az üzenet Pascalos felépítése:

```
type TMessage = Record
  Msg: Word;          //üzenetazonosító
  wParam: Word;             //Word típusú
  paraméter
  lParam: LongInt;  //LongInt típusú paraméter
  Result: LongInt;  //üzenet feldolgozásának
                            //eredménye
end;
```

Üzenetek feldolgozása

- Minden szál külön *üzenetsorral* rendelkezik.
- Az üzeneteket a Windows e sorokba helyezi, ezek alapján válogatja szét a több szálon futó alkalmazások közti eseményekhez tartozó üzeneteket.
- Minden szálban van ún. *üzenetkezelő ciklus* - a sorban álló üzeneteket kiolvassa, és továbbítja a megfelelő objektumhoz (ablakhoz).
- Minden ablak saját *ablakfüggvénnyel* rendelkezik, amiben feldolgozza a neki szánt üzeneteket.

Üzenetek feldolgozása

- Az ablakfüggvény egy nagy logikai elágazás:

```
Elágazás
  üzenet = billentyű lenyomása
     billentyű lenyomás feldolgozása
  üzenet = egér kattintás
     egér kattintás feldolgozása
  …
  Elágazás vége
```

II.2. Adattípusok az ObjectPascal-ban

- egyszerű típusok
  - sorszámozott típusok
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok
- összetett típusok
  - tömb
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

- **egyszerű típusok**
  - **sorszámozott típusok**
    - **EGÉSZEK**
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok
- összetett típusok
  - tömb
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

EGÉSZ TÍPUSOK

Típus	Értékkészlet	Formátum
Általános egész típusok a 32-bites Object Pascal implementációban		
Integer	-147483648...2147483647	előjeles, 32-bites
Cardinal	0…4294967295	előjel nélküli, 32-bites
Alapvető egész típusok		
Shortint	-128...127	előjeles, 8-bites
Smallint	-32768...32767	előjeles, 16-bites
Longint	-2147483648...214748364	előjeles, 32-bites
Int64	-2^63...2^63–1	előjeles, 64-bites
Byte	0…255	előjel nélküli, 8-bites
Word	0…65535	előjel nélküli, 16-bites
Longword	0…4294967295	előjel nélküli, 32-bites

Egész típusok műveletei

Aritmetikai műveletek	Jel, operandus	Eredmény
azonos előjel	**+** *egész*	egész
előjelváltás	**-** *egész*	egész
szorzás	*egész* ***** *egész*	egész
egész osztás	*egész* **Div** *egész*	egész
valós osztás	*egész* **–** *egész*	**valós** (= *real*)
maradékos osztás	*egész* **Mod** *egész*	egész
összeadás	*egész* **+** *egész*	egész
kivonás	*egész* **–** *egész*	egész
Logikai műveletek	**Jel, operandus**	**Eredmény**
bitenkénti NEM	**Not** *egész*	egész
bitenkénti ÉS	*egész* **And** *egész*	egész
bitenkénti VAGY	*egész* **Or** *egész*	egész
bitenkénti KIZÁRÓ VAGY	*egész* **Xor** *egész*	egész
bitenkénti léptetés balra n-szer	*egész* **Shl** *n*	egész
bitenkénti léptetés jobbra n-szer	*egész* **Shr** *n*	egész

Egész típusok műveletei

Összehasonlító műveletek	Jel, operandus	Eredmény
kisebb	*egész* **<** *egész*	Boolean
kisebb-egyenlő	*egész* **<=** *egész*	Boolean
egyenlő	*egész* **=** *egész*	Boolean
nagyobb-egyenlő	*egész* **>=** *egész*	Boolean
nagyobb	*egész* **>** *egész*	Boolean
nem egyenlő	*egész* **<>** *egész*	Boolean
Elemvizsgálat halmazban	**Jel, operandus**	**Eredmény**
halmaz tartalmazza-e az elemet	*egész* **In** *egészhalmaz*	Boolean

Egész típusok állandói, metódusai

- ***Előre deklarált egész állandók:***
  - **MaxInt** = 32 767
  - **MaxLongInt** = 2 147 483 647

- ***Egész típusokra alkalmazható, előre definiált metódusok:***
  - *Abs(), Dec(), Hi(), High(), Inc(), Lo(), Low(), Odd(), Ord(), Pred(), Random(), Read(), ReadLn(), SizeOf(), Sqr(), Str(), Succ(), Swap(), Val(), Write(), WriteLn()*.

LOGIKAI

- **egyszerű típusok**
  - **sorszámozott típusok**
    - egészek
    - **LOGIKAI TÍPUSOK**
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok

- összetett típusok
  - tömb
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

LOGIKAI TÍPUSOK

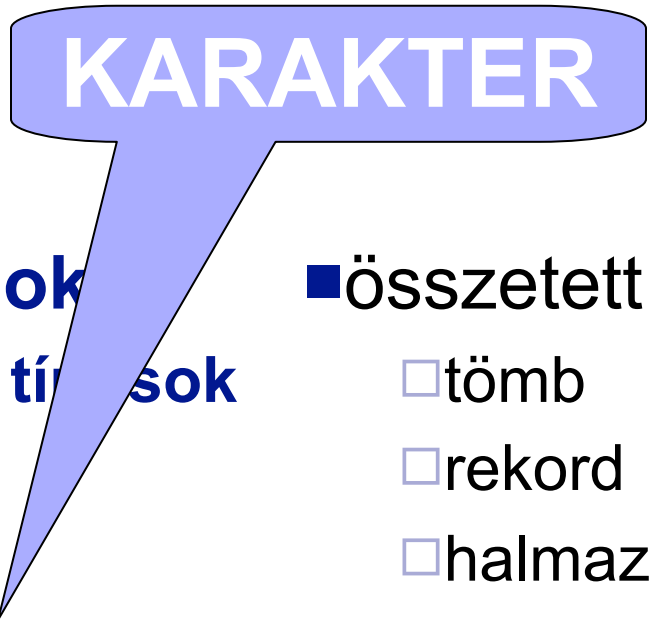
Típus	Értékkészlet	Formátum
Boolean	False, True	8 bit, 0: False / 1: True
ByteBool	False, True	8 bit, 0: False / nem 0: True
WordBool	False, True	16 bit, 0: False / nem 0: True
LongBool	False, True	32 bit, 0: False / nem 0: True

Előre deklarált logikai állandók: False, True

Előre definiált metódusok:
Dec(), High(), Inc(), Low(), Ord(), Pred(), SizeOf(), Succ(), Write(), WriteLn().

Logikai típusok műveletei

Logikai műveletek	Jel, operandus	Eredmény
logikai NEM	**Not** *logikai*	Boolean
logikai ÉS	*logikai* **And** *logikai*	Boolean
logikai VAGY	*logikai* **Or** *logikai*	Boolean
logikai KIZÁRÓ VAGY	*logikai* **Xor** *logikai*	Boolean
Összehasonlító műveletek	**Jel, operandus**	**Eredmény**
kisebb	*logikai* **<** *logikai*	Boolean
kisebb-egyenlő	*logikai* **<=** *logikai*	Boolean
egyenlő	*logikai* **=** *logikai*	Boolean
nagyobb-egyenlő	*logikai* **>=** *logikai*	Boolean
nagyobb	*logikai* **>** *logikai*	Boolean
nem egyenlő	*logikai* **<>** *logikai*	Boolean
Elemvizsgálat halmazban	**Jel, operandus**	**Eredmény**
halmaz tartalmazza-e az elemet	*logikai* **In** *logikaihalmaz*	Boolean



- **egyszerű típusok**
  - **sorszámozott típusok**
    - egészek
    - logikai típusok
    - **KARAKTER**
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok

- összetett típusok
  - tömb
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

KARAKTER TÍPUSOK

Típus	Értékkészlet	Formátum
Char	#0 - #255	8 bit, karakterkód

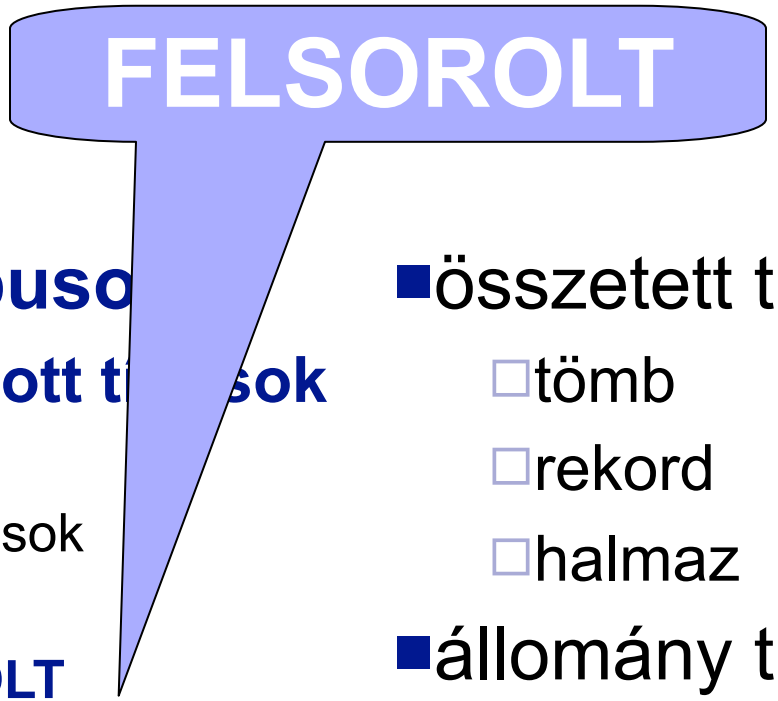
Karakter típusokon végezhető műveletek:

Összehasonlító műveletek	Jel, operandus	Eredmény
kisebb	*karakter* **<** *karakter*	Boolean
kisebb-egyenlő	*karakter* **<=** *karakter*	Boolean
egyenlő	*karakter* **=** *karakter*	Boolean
nagyobb-egyenlő	*karakter* **>=** *karakter*	Boolean
nagyobb	*karakter* **>** *karakter*	Boolean
nem egyenlő	*karakter* **<>** *karakter*	Boolean
Elemvizsgálat halmazban	**Jel, operandus**	**Eredmény**
halmaz tartalmazza-e az elemet	*karakter* **In** *karakterhalmaz*	Boolean

Karakter típusok állandói, metódusai

- ***Előre deklarált karakter állandók:***
  - - Aposztrófok között egy karakter – pl.: ’A’
  - - # jel után decimális karakterkód – pl.: #72
  - - #$ jel után hexadecimális karakterkód – pl.: #$2C
  - - ^ jellel vezérlőkarakter – pl.: ^G

- ***Előre definiált metódusok:***
  Chr(), Dec(), High(), Inc(), Low(), Ord(), Pred(), Read(), ReadLn(), SizeOf(), Succ(), UpCase(), Write(), WriteLn().



- **egyszerű típuso**
  - **sorszámozott t sok**
    - egészek
    - logikai típusok
    - karakter
    - **FELSOROLT**
    - intervallum
  - valós típusok
- karakterláncok

- összetett típusok
  - tömb
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

FELSOROLT TÍPUSOK

- ***Deklarálása:***

```
(értékazonosító [ , értékazonosító… ])
```

- ***Tárolása:*** 1 – 255 értékazonosító és *{$Z-}* esetén 1 bájton, 255-nél több értékazonosító vagy *{$Z-}* esetén 2 bájton történik.
- pl.:

```
Type Napok = (Hetfo, Kedd, Szerda,
Csutortok, …);
```

Felsorolt típusok műveletei

Összehasonlító műveletek	Jel, operandus	Eredmény
kisebb	*felsorolt* **<** *felsorolt*	Boolean
kisebb-egyenlő	*felsorolt* **<=** *felsorolt*	Boolean
egyenlő	*felsorolt* **=** *felsorolt*	Boolean
nagyobb-egyenlő	*felsorolt* **>=** *felsorolt*	Boolean
nagyobb	*felsorolt* **>** *felsorolt*	Boolean
nem egyenlő	*felsorolt* **<>** *felsorolt*	Boolean
Elemvizsgálat halmazban	**Jel, operandus**	**Eredmény**
halmaz tartalmazza-e az elemet	*felsorolt* **In** *felsorolthalmaz*	Boolean

Előre definiált metódusok:

Dec(), High(), Inc(), Low(), Ord(), Pred(), SizeOf(), Succ().

INTERVALLUM

- **egyszerű típusok**
  - **sorszámozott típusok**
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - **INTERVALLUM**
  - valós típusok
- karakterláncok

- összetett típusok
  - tömb
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

INTERVALUM TÍPUSOK

- Az intervallum egy már meglévő felsorolt típus alhalmazát jelenti.

  pl.: Felsorolt típus:

  ```
  type TSzinek = (Piros, Kek, Zold, Sarga,
  Lila, Bibor, Feher, Fekete);
  ```

  ennek egy intervalluma:

  ```
  type TSzinek = Zold..Feher;
  ```

  Itt a TSzinek a következő értékekből áll: Zold, Sarga, Lila, Bibor és Feher.

- ***Műveletek:*** az alaptípus műveletei.
- ***Állandók:*** az alaptípus állandói.
- ***Metódusok:*** az alaptípus metódusai.

Intervallum típus deklarálása

```
kezdőérték .. záróérték
```

- kezdőérték: egész, logikai, karakteres vagy felsorolt típusú állandó,
- záróérték: *kezdőérték* típusú, azzal egyenlő vagy annál nagyobb állandó.

{$R+}* esetén csak a deklarált intervallumba eső értékeket veheti fel az intervallum típusú változó, különben futási hiba ill. a *SysUtils* unit használata esetén **ERangeError** kizárás lép fel. *{$R+} esetén nincs intervallum ellenőrzés.

VALÓS

- **egyszerű típusok**
  - sorszámozott típus
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - **VALÓS TÍPUSOK**
- karakterláncok

- összetett típusok
  - tömb
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

EGYSZERŰ – VALÓS TÍPUSOK

Típus	Értékkészlet	Tárolás	Méret bájtokban
Real48	2.9 x 10^–39 .. 1.7 x 10^38	11–12	6
Single	1.5 x 10^–45 .. 3.4 x 10^38	7–8	4
Double	5.0 x 10^–324 .. 1.7 x 10^308	15–16	8
Extended	3.6 x 10^–4951 .. 1.1 x 10^4932	19–20	10
Comp	–2^63+1 .. 2^63 –1	19–20	8
Currency	–922337203685477.5808.. 922337203685477.5807	19–20	8

Valós típusok műveletei

Aritmetikai műveletek	**Jel, operandus**	**Eredmény**
azonos előjel	**+** *valós*	valós
előjelváltás	**-** *valós*	valós
szorzás	*valós* ***** *valós*	valós
valós osztás	*valós* **–** *valós*	**valós** (= *real*)
összeadás	*valós* **+** *valós*	valós
kivonás	*valós* **–** *valós*	valós
Összehasonlító műveletek	**Jel, operandus**	**Eredmény**
kisebb	*valós* **<** *valós*	Boolean
kisebb-egyenlő	*valós* **<=** *valós*	Boolean
egyenlő	*valós* **=** *valós*	Boolean
nagyobb-egyenlő	*valós* **>=** *valós*	Boolean
nagyobb	*valós* **>** *valós*	Boolean
nem egyenlő	*valós* **<>** *valós*	Boolean

Valós típusok műveletei

Abs(), ArcTan(), Cos(), Exp(), Frac(), Int(), Ln(), Pi(), Random(), Read(), ReadLn(), Round(), Sin(), SizeOf(), Sqr(), Sqtr(), Str(), Trunc(), Val(), Write(), WriteLn().

- Egy valós típus lebegőpontos ábrázolással megadható számokat határoz meg.

KARAKTERLÁNC

- egyszerű típusok
  - sorszámozott típusok
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- **KARAKTERLÁNCOK**
- összetett típusok
  - tömb
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

KARAKTERLÁNCOK

Típus	Maximális hossz	Szükséges memória	Alkalmazás
ShortString	255 karakter	2 – 256 bájt	kompatibilitás visszafelé
AnsiString	~2^31 karakter	4 bájt – 2 GB	8-bit (ANSI) karakterek
WideString	~2^30 karakter	4 bájt – 2 GB	Unicode karakterek; COM szerverek és interfészek

Karakterláncok deklarálása

`String``[ maximális hossz ]`

- A karakterláncok indexelhetőek, akár a tömbök. A 0. pozíción a karakterlánc aktuális hossza tárolódik, karakter típussal.

pl.:

```
var S : String[10];
S := 'Hello világ';
```

Ekkor az S[2] az 'e' karakter.

Karakterlánc állandói műveletei

- ***Előre deklarált karakter állandók:***
  - - Aposztrófok között egy karakter – pl.: ’A’
  - - # jel után decimális karakterkód – pl.: #72
  - - #$ jel után hexadecimális karakterkód – pl.: #$2C
  - - ^ jellel vezérlőkarakter – pl.: ^G
- ***Előre definiált metódusok:***
  - *Concat(), Copy(), Delete(), High(), Insert(), Length(), Low(), Pos(), Read(), ReadLn(), SizeOf(), Str(), Val(), Write(), WriteLn()*.

TÖMB

- egyszerű típusok
  - sorszámozott típusok
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok

- **összetett típusok**
  - **TÖMB**
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

STATIKUS TÖMB

Fix mérettel létrehozott, azonos típusok adathalmaza.

Deklarálása:

```
Array [ indextípus [ , indextípus] ] of
  alaptípus;
```

- indextípus: egyszerű típus, melynek értékkészlete nem haladja meg a 2 GB-ot.

Egy és többdimenziós tömb

- egydimenziós (vektor):
  - `var Vektor: array[1..100] of Char;`
  - hivatkozás egy elemre: Vektor[30]
- többdimenziós (mátrix):
  - `type TMatrix = array[1..10, 1..50] of Real;`
  - hivatkozás egy elemre: Matrix[5, 30] vagy Matrix [5],[30]

DINAMIKUS TÖMB

A dinamikus tömböknek nincs előre rögzített mérete vagy hossza. Ehelyett a memóriában csak később történik meg a helyfoglalása, amikor értéket rendelünk hozzá vagy átadjuk a **SetLength** eljárásnak.

```
var DinamikusVektor: array of Real;

SetLength(DinamikusVektor, 20);
```

Ez egy 20 valós elemből álló tömböt hoz létre, amely 0-tól 19-ig indexelt.

REKORD

- egyszerű típusok
  - sorszámozott típusok
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok

- **összetett típusok**
  - tömb
  - **REKORD**
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

REKORD

A rekord elemek heterogén halmazát jelenti. Az egyes elemeket mezőknek hívjuk. A rekord típus deklarációjában megadjuk valamennyi mező nevét és típusát.

Deklarálás:

```
type RekordNev = record
  Mezo_1: típus1;
  ...
  Mezo_n: típusn;
end;
```

- ahol:
- - RekordNev: egy alkalmas azonosító,
- - Mezo_1..n: alkalmas azonosítók,
- - típus1..n: ObjectPascal típusok.

Rekordok használata

```
type
  TAdatList = record
    Adat1: Integer;
    Adat2: String[20];
    Adat3: Boolean;
end;
```

Rekordmezők elérése:

változóazonosító.mezőazonosító

```
  var AktList: TAdatLista;
      …
      AktList.Adat1 := 125;
      AktList.Adat2 := 'Szöveg';
      AktList.Adat3 := True;
```

HALMAZ

- egyszerű típusok
  - sorszámozott típusok
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok

- **összetett típusok**
  - tömb
  - rekord
  - **HALMAZ**
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

HALMAZ

A halmaz azonos egyszerű, sorszámozott típusú értékek gyűjteménye.

Deklarálás:

```
Set of alaptípus;
```

- alaptípus: olyan sorszámozott típus, amelynek értékkészlete nem haladja meg a 0-255 tartományt.

```
type
THalmaz = 1..250;
THalm = set of THalmaz;
```

Halmazok műveletei

Halmaz műveletek	**Jel, operandus**	**Eredmény**
metszet	*halmaz* * *halmaz*	halmaz
unió	*halmaz* + *halmaz*	halmaz
különbség	*halmaz* - *halmaz*	halmaz
Összehasonlító műveletek	**Jel, operandus**	**Eredmény**
tartalmazás	*halmaz* <= *halmaz*	Boolean
egyenlő	*halmaz* = *halmaz*	Boolean
tartalmazás	*halmaz* >= *halmaz*	Boolean
nem egyenlő	*halmaz* <> *halmaz*	Boolean
Elemvizsgálat halmazban	**Jel, operandus**	**Eredmény**
halmaz tartalmazza-e az elemet	*érték* **In** *halmaz*	Boolean

SZÖVEGES ÁLLOMÁNY

- egyszerű típusok
  - sorszámozott típusok
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok

- összetett típusok
  - tömb
  - rekord
  - halmaz
- **állomány típusok**
  - **SZÖVEGES ÁLLOMÁNY**
  - típusos állomány
- mutatók
  - típusos mutató
- karakterlánc mutató

SZÖVEGES ÁLLOMÁNY

A szöveges állomány egy előre definiált állománytípus.
Típusazonosítója: **`TextFile`**.

Felépítése:

A szöveges állományt sorok alkotják, amelyeket a *kocsi vissza (= Carriage Return) / új sor (= New Line)* karakterpár zár: CR/LF (^M^J). A logikai állomány végét a *fájl vége (= End Of File)* karakter jelzi: EOF (^Z).

Deklarálás:

```
azonosító: TextFile;


AdatFile: TextFile;
```

Állománykezelés

Szöveges állomány csak változóként deklarálható.

- Fizikai állományhoz való hozzárendelés: **AssignFile**
- Megnyitás csak írásra: **Append** ill. **Rewrite**
- Megnyitás csak olvasásra: **Reset**
- Szekvenciális kiírás: **Write** ill. **WriteLn**
- Szekvenciális beolvasás: **Read** ill. **ReadLn**
- Állomány bezárása: **CloseFile**

TÍPUSOS ÁLLOMÁNY

- egyszerű típusok
  - sorszámozott típus
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok

- összetett típusok
  - tömb
  - rekord
  - halmaz
- **állomány típusok**
  - szöveges állomány
  - **TÍPUSOS ÁLLOMÁNY**
- mutatók
  - típusos mutató
- karakterlánc mutató

TÍPUSOS ÁLLOMÁNY

A típusos állomány azonos típusú elemek rendezett halmaza.

Deklarálás: `File Of alaptípus;`

```
AdatFile_1: File of Byte;
AdatFile_2: File of Array[1..20] of String
  [20];
AdatFile_3: File of TRecordSet;
```

Felépítése:

A típusos állományt a megadott alaptípusú elemek alkotják. Az első elem pozíciója *0*, a másodiké *1*, az n-ediké *n*. Az

Állománykezelés

Típusos állomány csak változóként deklarálható.

- Fizikai állományhoz való hozzárendelés: **AssignFile**
- Megnyitás csak írásra: **Append** ill. **Rewrite**
- Megnyitás csak olvasásra: **Reset**
- Szekvenciális kiírás: **Write** ill. **WriteLn**
- Szekvenciális beolvasás: **Read** ill. **ReadLn**
- Állomány bezárása: **CloseFile**
- Fizikai pozicionálás: **Seek**

MUTATÓK

- egyszerű típusok
  - sorszámozott típusok
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok

- összetett típusok
  - tömb
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- **mutatók**
  - **TÍPUSOS MUTATÓ**
- karakterlánc mutató

TÍPUSOS MUTATÓK

A mutató bármilyen típusra deklarálható, és egy memóriacímre mutat. Tárolása 32 biten történik: *szegmens:eltolási cím*.

Ha a mutató nem tartalmaz konkrét címet, akkor az értéke: **nil**.

Deklarálása: `^alaptípus`

```
type
  ByteMutato = ^Byte;
  TTomb = Array[1..10] of Integer;
  PTomb = ^TTomb;
```

Mutatók műveletei

Összehasonlító műveletek	Jel, operandus	Eredmény
egyenlő	*mutató* **=** *mutató*	Boolean
nem egyenlő	*mutató* **<>** *mutató*	Boolean

KARAKTERLÁNC MUTATÓK

- egyszerű típusok
  - sorszámozott típu
    - egészek
    - logikai típusok
    - karakter
    - felsorolt
    - intervallum
  - valós típusok
- karakterláncok

- összetett típusok
  - tömb
  - rekord
  - halmaz
- állomány típusok
  - szöveges állomány
  - típusos állomány
- mutatók
  - típusos mutató
- **KARAKTERLÁNC MUTATÓ**

KARAKTERLÁNC MUTATÓK

Típusazonosítója: **PChar**. Tárolása 32 biten történik: *szegmens:eltolási cím*.

Ha a mutató nem tartalmaz konkrét címet, akkor az értéke: **nil**.

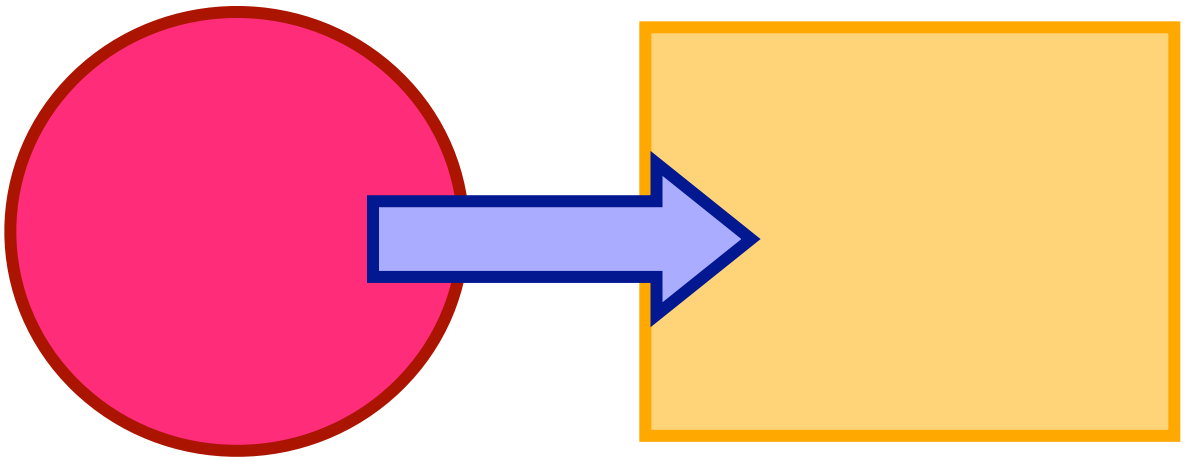
A **PChar** típust a *System unit* deklarálja **^Char** típusként.

```
var P: PChar;
P := 'Hello világ!';
```

Karakterlánc mutatóknműveletei

Címkezelő műveletek	Jel, operandus	Eredmény
eltolási cím növelése	*mutató* **+** *egész*	*mutató*
eltolási cím csökkentése	*mutató* **–** *egész*	*mutató*
eltolási címek különbsége	*mutató* **-** *mutató*	*mutató*
Összehasonlító műveletek	**Jel, operandus**	**Eredmény**
kisebb	*mutató* **<** *mutató*	Boolean
kisebb-egyenlő	*mutató* **<=** *mutató*	Boolean
egyenlő	*mutató* **=** *mutató*	Boolean
nagyobb-egyenlő	*mutató* **>=** *mutató*	Boolean
nagyobb	*mutató* **>** *mutató*	Boolean
nem egyenlő	*mutató* **<>** *mutató*	Boolean

Típuskonverziók az Object Pascalban



Chr(X: Byte): Char;

Karakterkódhoz tartozó karaktert ad vissza.

```
Chr(65); → 'A'
```

Int(X: Extended): Extended;

Valós típus egész részét adja vissza.

```
Int(10.125); → 10.0
```

Frac(X: Extended): Extended;

Valós típus tizades részét adja vissza.

```
Frac(10.125); → 0.125
```

Round(X: **Extended**): **Int64**;

Valós típust a legközelebbi egészre kerekíti, és *Int64* típusként adja vissza. Ha a tizedesrész 0.5, akkor a 0-tól távolabbi egészre kerekít.

```
Round(10.49); → 10
Round(10.51); → 11
Round(10.5); → 11
Round(-10.5); → -11
```

Trunc(X: **Extended**): **Int64**;

Valós típus egész részét *Int64* típusként adja vissza.

```
Trunc(10.49); → 10
Trunc(-10.49); → -10
```

Str(X:**Real** [:mezőszélesség [:tizedesszám]]; **var** S: **String**);

Egész vagy valós értéket karakterlánccá alakít.

Val(S:String; var V:Real; var Code: Integer);

Egész vagy valós értékké alakít egy karakterláncot. Ha V típusa szerint ez nem lehetséges, akkor a *Code*-ba az első hibás karakter indexe kerül, különben 0.

```
var x: LongInt;
    y: LongInt;
Val('256',x,y); → x=256, y=0
Val('25a6',x,y); → x=0, y=3
```

IntToHex(Value: Integer; Digits: Integer): string;

A *Value* érték *Digits* számjegy hosszúságúra alakított hexadecimális szöveges formáját adja vissza.

IntToStr(Value: Integer): string;

Egész típust szöveggé alakít.

StrToInt(Const S: string): Integer;

Karakterlánc egész típusú értékét adja vissza.

DateToStr(Date:TDateTime): string;

Dátumot szöveggé alakít.

StrToDate(const S: string): TDateTime;

Karakterláncot dátum formátummá alakít.

FloatToStr(Value: Extended): string;

15 szignifikáns jegyen karakterlánccá alakítja a valós típust.

FormatFloat(const Format: **string**; **Value**: **Extended**): **string**;

Formázott karakterlánccá alakítja a valós típust.

```
const x = 123456.789
FormatFloat('##########.00', x); );
     → x='123456.79'
```

StrToFloat(const S: **string): Extended;**

A karakterláncként tárolt, *[+/-]ddd.ddd…[E +/-ddd]* formátumú valós értéket adja vissza.

II.3. Objektumorientált programozás (OOP)

Az objektumorientált programozás nagyon hasonlít a **modellezésre**, amit

- az absztrakció,
- a megkülönböztetés,
- az osztályozás,
- az általánosítás és a specializálás
- a kapcsolatok felépítése
- a részekre bontás jellemez.

Egy objektumorientált program egymással kommunikáló objektumokból áll, ahol minden egyes objektum jól meghatározott feladattal rendelkezik.

Fogalmak

- **Absztrakció**

  Az absztrakció során a valós világ objektumainak csak a számunkra fontos jellegzetességeit vesszük figyelembe.

  Egy tartószerkezetet nem úgy ábrázolunk a tervrajzokon, mintha egy fénykép lenne, hanem vonalakkal az alakját, csomópontjait, méretszámokkal a méreteit, stb., amik az elkészítéséhez feltétlenül szükségesek.

- **Megkülönböztetés**

  Az objektumokat a fontos tulajdonságaik (attribútumaik) alapján különböztetjük meg egymástól.

  szórt likacsú fafajok ↔gyűrűs likacsú fafajok

Fogalmak – folyt.

- **Osztályozás**

  Az objektumokat bizonyos tulajdonságaik alapján rendszerezzük, osztályokba soroljuk.

  „tölgy, bükk” → kemény lombosok

  „nyár, hárs” → lágy lombosok

  „erdeifenyő, vörös fenyő” → fenyők

- **Általánosítás, specializálás**

  Általánosítás: Olyan folyamat, amelynek során több objektum leírásából kiemeljük a közös jellemzőket.

  gyűrűs likacsú → lombos → fa → növény

  Specializálás: Olyan folyamat, amelynek során egy objektum leírásához további, egyedi jellemzőket adunk hozzá.

Fogalmak – folyt.

- **Kapcsolatok felépítése és részekre bontás**

  Két objektum kapcsolata lehet *ismeretségi* (= együttműködési kapcsolat) vagy *tartalmazási* (= egész-rész kapcsolat).

  Ismeretségi kapcsolat esetén a két objektum egyike sem függ a másiktól, egymástól függetlenül léteznek.

  Tartalmazási kapcsolat esetén az egyik objektum része a másiknak: ha a tartalmazó objektum megszűnik, akkor megszűnik a tartalmazott is.

Fogalmak – folyt.

- **Üzenet**

  Az objektumokat üzenetek révén kérhetünk meg bizonyos feladatok elvégzésére.

  A megszólított objektumnak ismernie kell az adott feladatot és annak az alkalmazási feltételeit is.

- Például a megszólított objektum a *villanyszerelő*, az üzenet a *szerel* ill. a *főz*, amely egy paraméterrel is rendelkezik *vízvezeték/villany/főz*:

`villanyszerelő.szerel(vízvezeték)` → paraméterhiba

`villanyszerelő.szerel(villany)` → végrehajtás

`villanyszerelő.főz(ebéd)` →

Fogalmak – folyt.

- **Felelősség**

  Valamennyi objektum jól meghatározott feladattal rendelkezik, amelynek elvégzéséért felelős.

  Ez azt jelenti, hogy mindegy milyen módszerrel alakítja ki a villamoshálózatot a villanyszerelő, az a fontos, hogy rendben elkészüljön.

- **Bezárás, információ elrejtése**

  A feladatok megoldása csak a feladatot végző objektumra tartozik. Az objektum belügyeibe nem avatkozhatunk be, az objektummal kizárólag az ún. interfészen keresztül lehet kommunikálni.

  Az objektum tehát azokat az információkat, amelyek más objektumok számára nem fontosak, elrejti.

Fogalmak – folyt.

- **Osztályozás**

  Az objektumoknak példányai vannak, amelyek az általános objektum megvalósulásai. Ezeket a példányokat a viselkedésük alapján osztályokba soroljuk.

  - Pl.: a villanyszerelőre is több osztályt állapíthatunk meg:

villanyszerelő, kezdő villanyszerelő, mester villanyszerelő, stb.

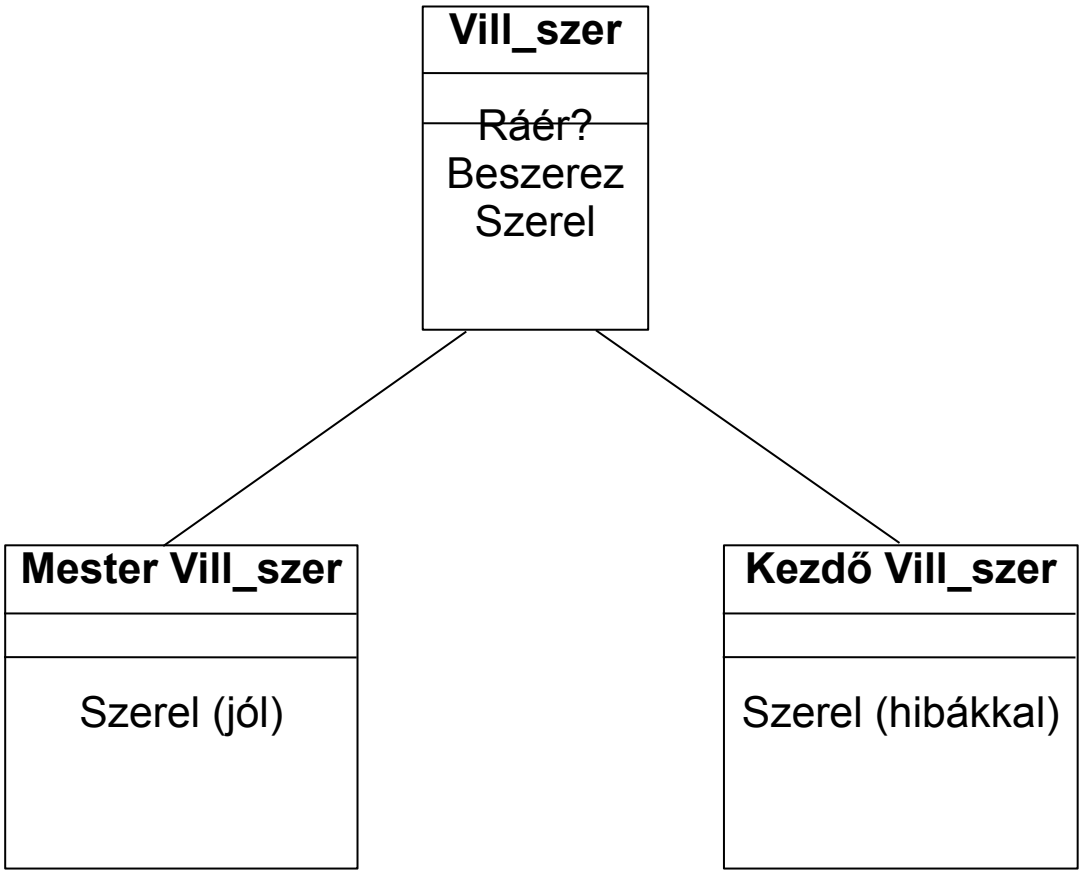
- **Öröklődés (= inheritance)**

  Egy osztály tulajdonságokat és viselkedésformákat örökölhet egy másik osztálytól. Ez tulajdonképpen specializálás. Az utód osztályban csak az ős osztálytól való eltéréseket kell megadni.

Fogalmak – folyt.

Ős osztályok

Utód osztályok



Fogalmak – folyt.

- **Polimorfizmus (= többalakúság)**

  Különböző objektumok különbözőképpen reagálhatnak egyazon üzenetre.

  - Pl.: a *szerel* üzenet hatására a villanyszerelő az elektromos hálózaton fog dolgozni, míg a vízvezeték szerelő a vízrendszeren.

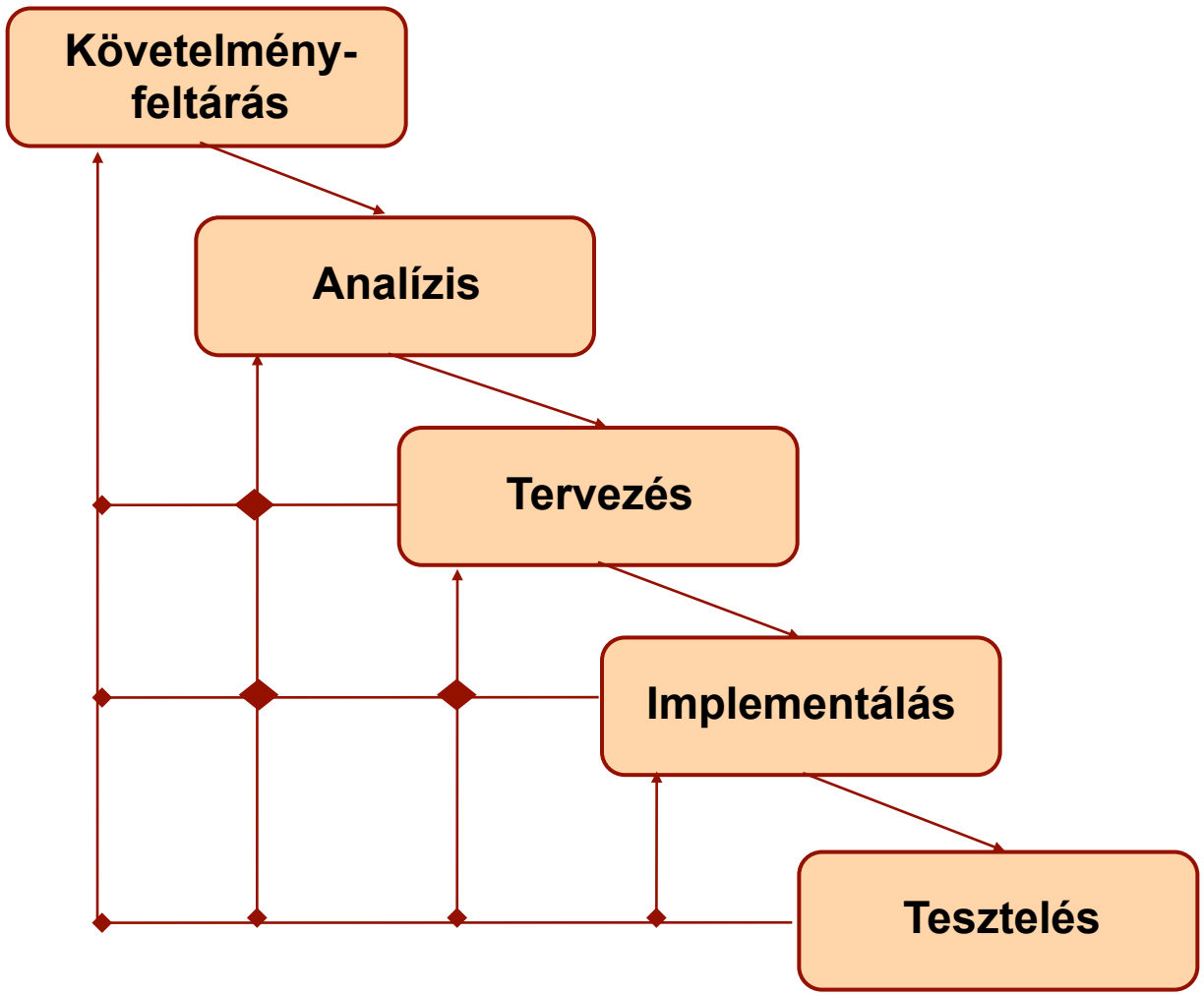
```
villanyszerelő.szerel()      vízvezetékszerelő.szerel()
```

- **Futás alatti kötés**

  Programozáskor nem lehet minden esetben tudni, hogy amikor megszólítunk egy másik objektumot, akkor ténylegesen milyen műveletek hajtódnak végre.

  Bizonyos műveletekről csal futási időben derül ki, hogy melyikre is van szükség. Ezek a műveletek tehát csak

Az OOP programfejlesztés



Követelményfeltárás

- program készítésére felvett megrendelésnél össze kell gyűjteni a feladatmegoldásához szükséges információkat:
  - a program feladatát, célját,
  - a felhasználó igényeit.
- követelményfeltárás: a készítendő programmal szemben támasztott elvárások összegyűjtése.
- dokumentációja: a ***feladatspecifikáció***.

Analízis

- a fejlesztő együtt dolgozik a megrendelővel, hisz ő ért legjobban a problémakörhöz
- meg kell fontolni, hogy:
  - a feladat megoldható-e,
  - milyen eszközökkel oldható meg.
- készíteni kell egy ***objektum/adatmodell***t
- össze kell gyűjteni a legfontosabb ***használati esetek***et,

Tervezés

- az előzőekben kialakított modelleket, elképzeléseket kell továbbfejleszteni és részletezni
- ki kell dolgozni az egységek – objektum osztályok – feladatait, az implementációhoz szükséges adatszerkezeteket és algoritmusokat.
- a tervezési munkafolyamat dokumentációja a ***programterv***, amellyel a probléma lényegében megoldottnak tekinthető

Kódolás (implementálás)

- a kódolás a terv gyakorlati megvalósítása
- ennek a szakasznak az eredménye a ***forrásprogram*** vagy ***forráskód*** (= *source code*)

- A kód legyen:
  - jól áttekinthető, hogy a hibákat könnyebben kiszűrhessük,
  - célszerű megfelelő magyarázatokkal ellátni,
  - az egyes részfeladatok megoldását külön egységekbe kell megírni (újra felhasználhatóság)

- A Delphi nagy előnye, a rengeteg modul, ún. komponens (*VCL – Visual Component Library*)

Néhány aranybánya

- Torry's Delphi Page: www.torry.net
- Delphi Super Pages: delphi.icm.edu.pl
- Richey's Delphi-Box: inner-smile.com
- SwissDelphiCenter: www.swissdelphicenter.ch
- Programmer's Heaven: www.programmersheaven.com/zone2

Tesztelés

- A programot a lehető legteljesebb mértékben ellenőrizni kell, és ki kell küszöbölni minden hibát.
  - A *szintaktikai hibák* jó része kiderül már a forráskód fordításakor
  - A *szemantikai hibák* miatt a program nem az elvárások szerint működik, vagy akár sehogyan sem működik. Ezeknek a hibáknak a forrását megtalálni már nehezebb.
- A hibákat megtalálásához és kijavításához szükség van:
  - gondosan összeállított tesztadatokra,
  - számos különböző helyzet előidézésére.
- Fontos a felhasználókkal való kapcsolattartás, akik informálhatnak a problémákról.

A felhasználó mindig tud olyan helyzetet produkálni, amelyre a legrutinosabb programozó sem gondol!

Dokumentálás

- **Fejlesztői dokumentációk**:
  - analízismodell a feladatspecifikációval, szakterületi modellel és a használati esetekkel,
  - tervezési modell a program-, lista- és képernyőtervekkel,
  - a forrásprogram,
  - a tesztadatok és eredmények.
- **Felhasználói dokumentációk**:
  - a feladat leírása,
  - szükséges hardver- és szoftverkörnyezet,
  - a program telepítése, elindítása,
  - a program használata,
  - hibaüzenetek, hibaelhárítás.

II.4. Objektumok és osztályok

- Osztály
- Objektum
- Objektum létrehozása, felszabadítása
- Információ elrejtése és bezárás
- Objektumok kapcsolatai

Osztály

- Az **osztály** egy felhasználó (programozó) által meghatározott adattípus, amelynek van állapota (megjelenése), és vannak műveletei (viselkedése). Az osztály rendelkezik belső adatokkal és metódusokkal (eljárások és függvények), és egymásra hasonlító dolgok viselkedését és tulajdonságait írja le. Az osztály alapján példányokat (objektumokat) lehet létrehozni.

Osztály létrehozása

```
type
  TUjOsztaly = class
     Adat1, Adat2: Integer;
     procedure Eljaras1(a1, a2: Integer);
     function Fuggveny1: Boolean;
end;
```

Az eljárás és a függvény abban tér el egymástól, hogy a függvénynek mindig van valamilyen visszatérési értéke.

Objektum

- Az **objektum** az osztály egy példánya, vagyis az adott osztály által definiált típusú változó.
  - Ezek egyedek, amelyek futási időben memóriaterületet foglalnak le.
  - Az objektum az osztályában megadott típusú adatokkal dolgozik az osztályában megadott metódusok révén.
- Az objektumnak van **állapot**a, amelyet a pillanatnyi értékek képviselnek.
- Minden objektum egyértelműen azonosítható, és ez az **azonosság** független a mindenkori tárolt értéktől.

Objektumok létrehozása

- Az objektumokat létre kell hozni, és inicializálni kell.
- Amikor már nincs többé szükség rá, akkor meg kell semmisíteni, fel kell szabadítani az általa elfoglalt memóriaterületet.
  - Az objektum létrehozása: a **create** metódussal.
  - Ezt követően, azonnal inicializálni kell, azaz meg kell adni a kezdő adatait, és végre kell hajtani azokat a metódusait, amelyek a megfelelő működéséhez elengedhetetlenek. Ezt az inicializálást végző metódust nevezzük **konstruktornak**.
  - Az objektumot, miután ellátta a rá szabott feladatot, fel kell szabadítani. Ezt a feladatot a **destruktor** metódus végzi.
- Már létező komponensek (objektumok) esetén az objektum konstruktora és destruktora már készen van, egy általunk készített objektum esetében azonban nekünk kell megírnunk.

Objektum létrehozása Delphiben

```
var                                    //változók definiálása
Objektum1: TUjOsztaly;    //TUjOsztaly típusú
                                  Objektum1 nevű
  változó
Fugg1: Boolean;                        //Boolean típusú
  Fugg1                                      nevű változó
begin
Objektum1 := TUjOsztaly.Create; //az objektum
  létrehozása
Objektum1.Eljaras1(15, 32);
Fugg1 := Objektum1.Fuggveny1;
Objektum1.Free;                                  //az
  objektum {vagy Objektum1.Destroy;}
     megszűntetése
```

Konstruktor

Konstruktort készíteni abból áll, hogy az inicializáló eljárást a `procedure` helyett a `corstructor`-ral vezetjük be:

```
type
      TUjOsztaly = class
      Adat1, Adat2: Integer;
      constructor Init(a1, a2: Integer);
      procedure Eljaras1(a1, a2: Integer);
      function Fuggveny1: Boolean;
end;


constructor TUjOsztaly.Init(a1, a2: Integer);
begin
      Adat1:= a1;
      Adat1:= a1;
end;
```

Információ elrejtése, bezárás

- Az objektum a hozzá tartozó **adatokat és metódusokat egységbe zárja**, és a **külvilág** (más objektumok, külső metódusok) **elől** a felesleges információkat **elrejti**.

- Az Object Pascal nyelvben három hozzáférést jelölő kulcsszó, ill. hozzájuk tartozó hozzáférési lehetőség van:
- **private** (= *privát*): az osztály vagy objektum olyan adatit és metódusait jelöli, amelyeket csak az osztályon/ objektumon belülről lehet elérni, kívülről nem.
- **public** (= *nyilvános*): az itt deklarált adatokat és metódusokat bárhonnan el lehet érni. Ez tulajdonképpen nyilvános interfész.
- **protected** (= *védett*): a védett metódusokat és adatokat csak az osztályból, valamint az abból származtatott alosztályokból lehet elérni.

Objektumok kapcsolatai

- **Ismeretségi kapcsolat:**
  - Két objektum ismeretségi kapcsolatban van egymással, ha azok léte független egymástól, és legalább az egyik használja a másikat.
- **Tartalmazási kapcsolat:**
  - Két objektum tartalmazási kapcsolatban áll egymással, ha az egyik objektum fizikailag tartalmazza a másikat. Ha a tartalmazó megszűni, akkor vele együtt szűnik meg a tartalmazott objektum is. Ha a tartalmazott objektum nem vehető ki a tartalmazóból, akkor a kapcsolat erős, különben gyenge.

Osztályok kapcsolatai

- **Egy-egy kapcsolat:** az egyik osztály egy példánya a másik osztály legfeljebb egy példányával állhat kapcsolatban, és viszont.
- **Egy-sok kapcsolat:** az egyik osztály egy példánya egy másik osztály sok példányával állhat kapcsolatban, ugyanakkor a másik osztály egy-egy példánya az első osztály legfeljebb egy példányával állhat kapcsolatban.
- **Sok-sok kapcsolat:** mindkét osztály bármely példánya a másik osztály sok példányával állhat kapcsolatban.

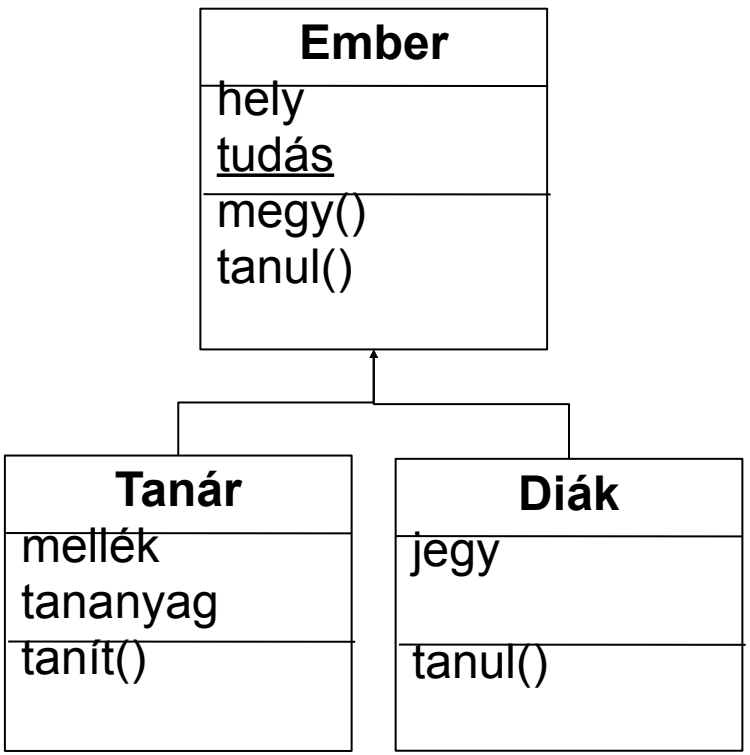
II.5. Öröklődés

- Az öröklődéssel egy specializációt végzünk, vagyis egy már létező osztály (**ős osztály**) tulajdonságait, metódusai felhasználva új osztályt (**utód osztály**) készítünk. Az utód örökli az ős osztály adatait és metódusait, amelyekhez újakat adhatunk ill. módosíthatjuk a már létezőket:
  - az ős osztályhoz új változókat adunk,
  - az ős osztályhoz új metódusokat adunk,
  - átírjuk az ős osztály metódusait.
- Az öröklés több szintű is lehet, a hierarchia legfelső osztályát **alaposztály**nak nevezzük.

Az öröklés szabályai

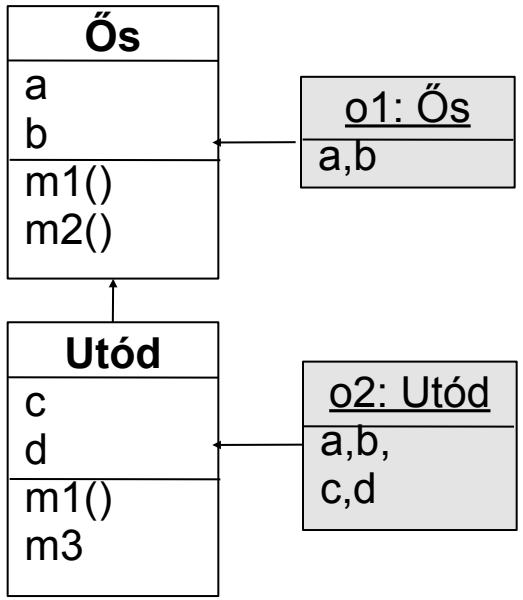
- egy osztályból több utód osztály is származhat,
- tetszőleges mélységű hierarchia hozható létre, de 5-nél több nem ajánlott,
- az öröklés tranzitív: ha A-t örökli B, és B-t örökli C, akkor C is örökli A-t.
- példányadatok öröklése: ős osztály adatai + saját (utód) adatok = utód osztály példányadatai,
- példánymetódusok öröklése: bármely metódus felülírható az utódban,
- az utód örökli az ős osztály kapcsolatait is

Öröklés osztálydiagramja



Üzenetküldés

- Az öröklés legfontosabb előnye, hogy az utód osztálynak adott üzenetek fogadásakor mindig a legutoljára definiált metódus hajtódik végre.



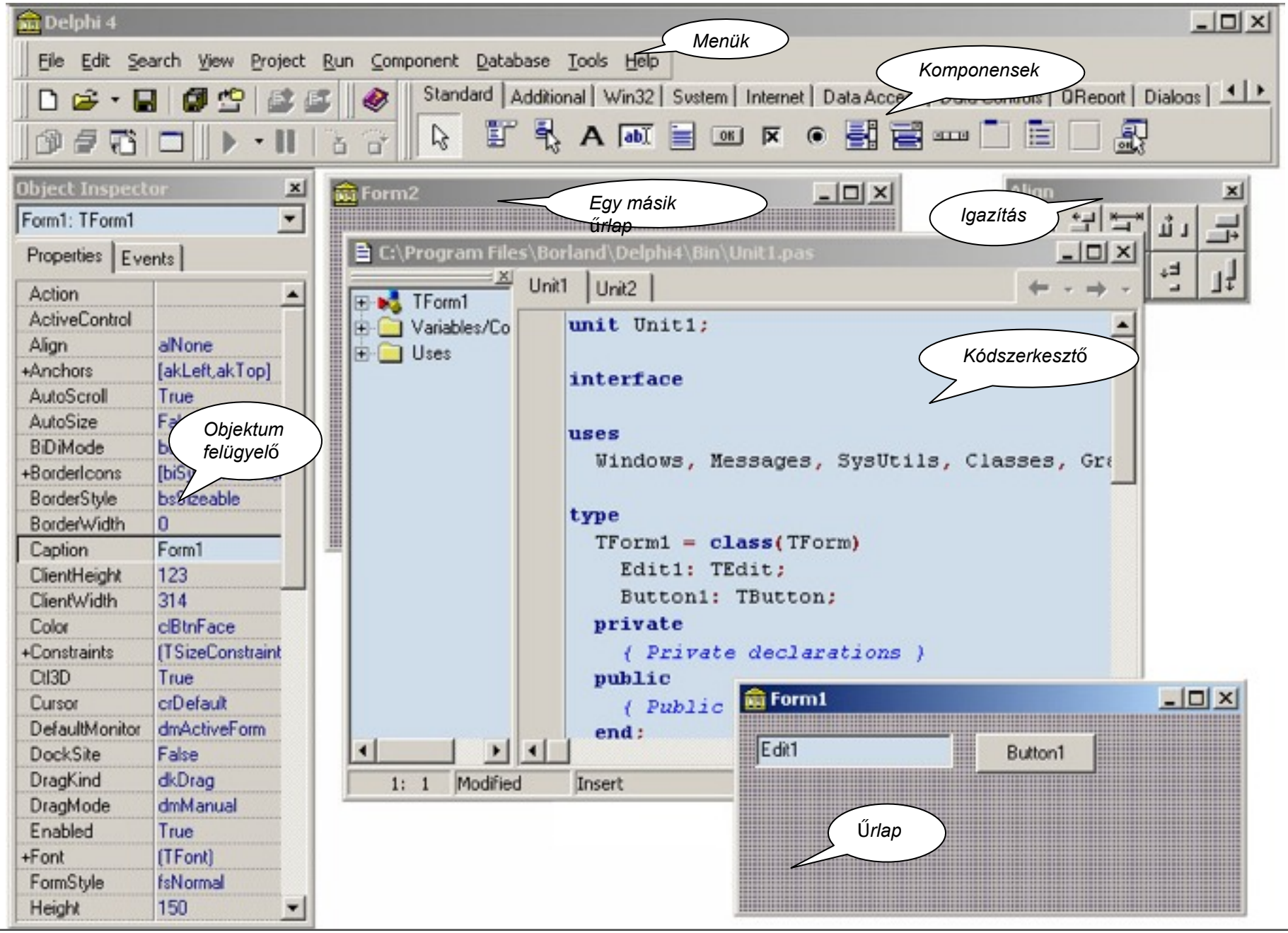
Küldhető üzenet		Végrehajtódó metódus
`o1.m1`	→	`Ős.m1`
`o1.m2`	→	`Ős.m2`
`o2.m1`	→	`Utód.m1`
`o2.m2`	→	`Ős.m2`
`o2.m3`	→	`Utód.m3`

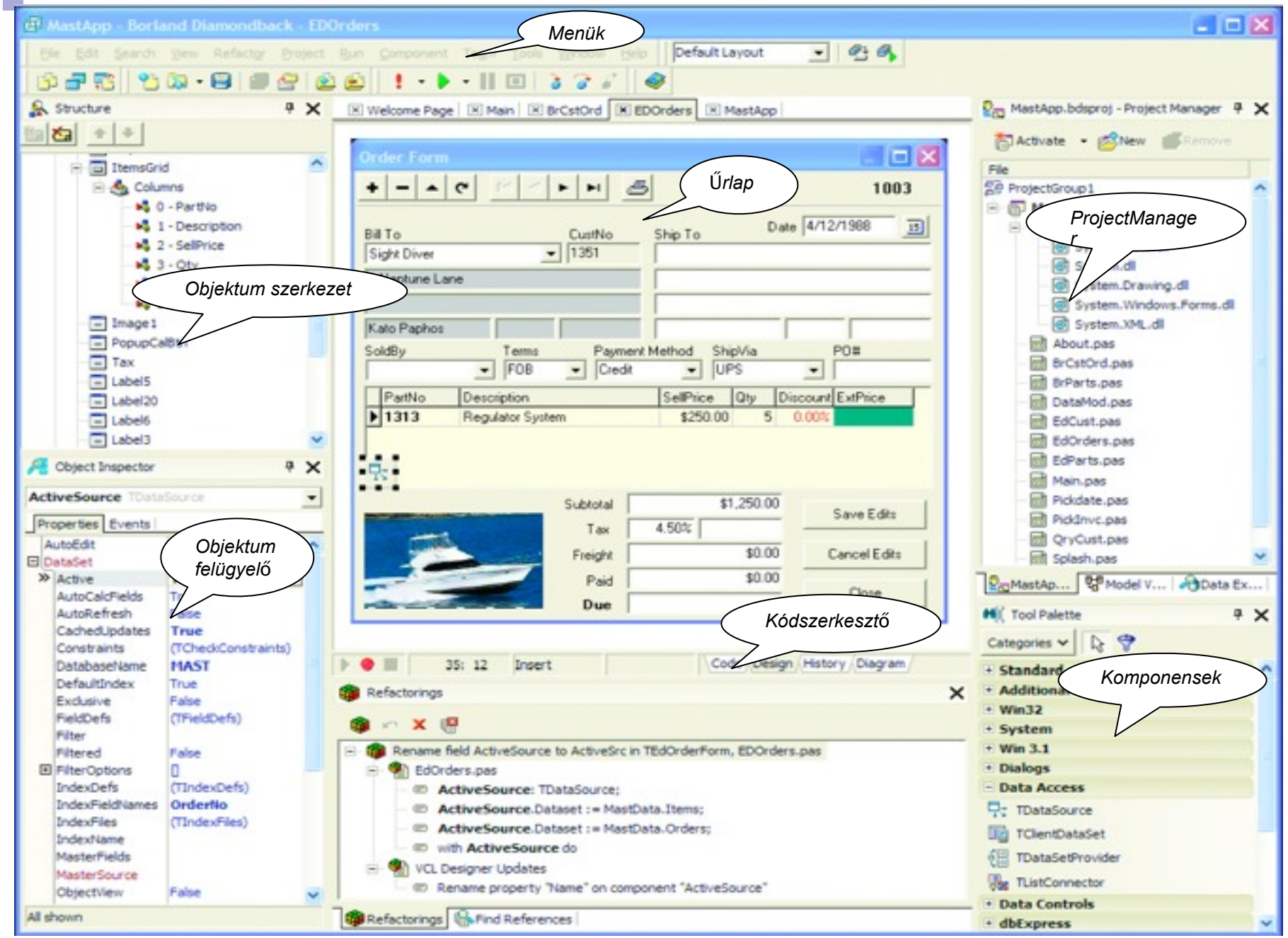
Láthatóság

- Egy osztály adatainak ill. metódusainak háromféle léthatóságot adhatunk:
  - **Nyilvános** (= *public*): minden kapcsoláatban álló kliens látja és használhatja.
  - **Védett** (= *protected*): hozzáférés csak öröklésen keresztül lehetséges.
  - **Privát** (= *private*): csak az osztály saját metódusai láthatják, használhatják.

Delphi alapok

- Delphi környezet
- Delphi alkalmazások felépítése
  - Projektállomány
  - Űrlapállomány
  - Egység állomány
  - Futtatható állomány


Delphi 4
File Edit Search View Project Run Component Database Tools Help
Menük
Komponensek
Standard Additional Win32 System Internet Data Acce QReport Dialogs
Object Inspector
Form1: TForm1
Properties Events
Action
ActiveControl
Align alNone
+Anchors [akLeft,akTop]
AutoScroll True
AutoSize
BiDiMode
+BorderIcons
BorderStyle
BorderWidth 0
Caption Form1
ClientHeight 123
ClientWidth 314
Color clBtnFace
+Constraints (TSizeConstraint
Ctl3D True
Cursor crDefault
DefaultMonitor dmActiveForm
DockSite False
DragKind dkDrag
DragMode dmManual
Enabled True
+Font (TFont)
FormStyle fsNormal
Height 150
Objektum felügyelő
Form2
Egy másik űrlap
Igazítás
C:\Program Files\Borland\Delphi4\Bin\Unit1.pas
Unit1 Unit2
TForm1
Variables/Co
Uses
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Classes, Gr
type
TForm1 = class(TForm)
Edit1: TEdit;
Button1: TButton;
private
{ Private declarations }
public
{ Public
end;
Kódszerkesztő
1: 1 Modified Insert
Form1
Edit1
Button1
Űrlap


Menük
Űrlap
ProjectManager
Objektum szerkezet
Objektum felügyelő
Kódszerkesztő
Komponensek

Delphi kezelőfelülete

- a **kezelőpanel**, amelyen a menüsor és a fájlkezeléshez, projektkezeléshez szükséges eszköztár található, ill. a korábbi verziók esetén a komponensek is itt szerepeltek;
- az **űrlap** (= *Form*), amely a tulajdonképpeni megszerkesztendő programfelületet, a Windows-ban megjelenő ablakokat jelenti;
- a **kódszerkesztő**, ahol az űrlaphoz tartozó forráskódot szerkeszthetjük;
- az **Objektum felügyelő** (= *Object Inspector*), ahol az űrlapon elhelyezett objektumok tulajdonságait állíthatjuk be, valamint az eseményekhez rendelhetünk eljárásokat;
- emellett számos hasznos ablak kapcsolható be, mint pl.:
  - a **Project Manager**, amely a projekthez tartozó űrlapokat és egységeket jeleníti meg,
  - az **üzenetek** (= *Messages*) ablak, ahol a fordító a hibákra, figyelmeztetésekre hívja fel a figyelmet,
  - a **figyelő lista** (= *Watch list*), ahol a futás közben követhetjük nyomon egyes változók mindenkori értékeit,
  - a projekt **objektum szerkezet**e (*Structure*).

Alkalmazások (projektek) felépítése

- Alkalmazások négy főrészből épülnek fel:
  - a projektállományból,
  - az egységekből (Object Pascal kód),
  - a hozzájuk tartozó Delphi erőforrásokból (űrlapok),
  - külső erőforrásokból.
- Új alkalmazás létrehozásakor a Delphi automatikusan generál egy projektállományt (= *project*), egy űrlapot (= *form*) és a hozzá tartozó egységet (= *unit*).

Projektállomány:
*.DPR – *Delphi Project*

programfej

hivatkozási rész

végrehajtó rész

```
program ElsoAlk;

uses
  Forms,
  Elso in 'Elso.pas' {MainForm};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.Run;
end.
```

projekt- és futtatható állomány neve

űrlapok és a hozzájuk tartozó egységek

fordítási direktíva

MainForm létrehozása

alkalmazás futtatása

Űrlapállomány:
*.DFM – *Delphi Form*

- Egy-egy alkalmazásban egy vagy több ablaka - űrlap (= *form*) van.
- Az űrlapok viselkedését az űrlapéval azonos nevű egységek (= *unit*) írják le.
- Az űrlapokon helyezhetjük el az alkalmazás funkciójának megfelelő gombokat, beviteli mezőket, listákat, táblázatokat, stb.
- Az így elkészített űrlapokat a Delphi bináris állományban tárolja.

Egység állomány: *.PAS

- Az űrlapok viselkedését az azonos elnevezésű egységállományok tartalmazzák - Delphi a közös név alapján pontosan tudja, hogy melyik ablaknak hogyan kell működnie.
- A Delphi a fordítás (*compile*) során
  - a PAS kiterjesztésű egységekből DCU (*Delphi Compiled Unit*) állományt hoz létre,
  - a szerkesztő (*linker*) a DFM és a DCU állományokat az EXE állományba szerkeszti
  - a RES külső erőforrásokkal együtt
  - a DPR alapján.

Egységfej

Illesztő rész

Kifejtő rész

Inicializációs rész (üres)

```
unit Elso;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls;

type
  TMainForm = class(TForm)
    BeEdit: TEdit;
    VezerloBtn: TButton;
    KiLabel: TLabel;
    procedure VezerloBtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.VezerloBtnClick(Sender: TObject);
begin
     KiLabel.Caption:=BeEdit.Text;
end;

end.
```

Delphi könyvtár egységek

`TMainForm` osztály deklarációja az általa tartalmazott objektumokkal, eljárásokkal

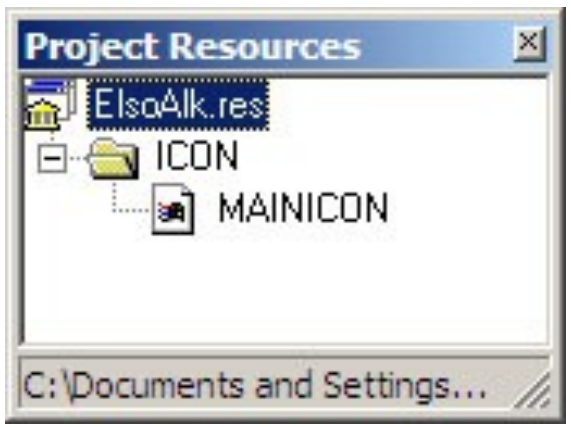
`MainForm` példány deklarációja

`TMainForm` osztály kifejtése: eljárások kódolása

Külső erőforrások:

*.RES, *.ICO, *.CUR, *.ANI, *.HLP, *.BMP, stb.

- Ezek olyan külső eszközök, amelyeket az EXE állomány mellett külön tárolunk, és a program futás közben nyitja meg és használja őket.
  - lehetnek saját készítésű egérmutatók (CUR, ANI), képek (BMP, GIF), súgó (HLP) vagy az alkalmazás ikonja (ICO).
- Ugyanakkor lehetőség van ezek egybeszerkesztésére is egy RES állományba, amelyet a programszerkesztő (= *linker*) beleépít majd a futtatható állományba.



Futtatható állomány: *.EXE

- A Delphi fordítója és programszerkesztője a projektállományból (DPR), az egységekből (PAS-DCU), a Delphi erőforrásokból (DFM), valamint az erőforrásokból (RES) futtatható állományt készít (EXE).