Objektum orientált programozás
(Object Oriented Programming = OOP)

Ajánlott irodalom:

Angster Erzsébet: Az objektumorientált tervezés és programozás alapjai

Dr. Kondorosi Károly, Dr. László Zoltán, Dr. Szirmay-Kalos László: Objektumorientált szoftverfejlesztés

Moduláris programozás

A modulokon belüli erős összetartás, a modulok között laza kapcsolat.

A modulok szintaktikailag jól elkülöníthetőek

Modulok közötti kapcsolatok száma minél kevesebb

Kicsi interfésze legyen a moduloknak

Az interfész legyen egyértelmű és jól definiált

Minél nagyobb része legyen zárt és sérthetetlen

Struktúrált programozás:

Program = adat + algoritmus

60-as években Böhm és Jacopini sejtése: bármely algoritmus leírható az alábbi 3 vezérlési szerkezet véges sokszori alkalmazásával:

szekvencia

szelekció

iteráció

A fenti sejtést Mills bizonyította

Struktúrált programozás:

A legkisebb modul az eljárás

Az eljárások adatokon dolgoznak, a zártság érdekében megpróbáljuk a globális adatokat minimalizálni.

A lokális adatok elvesznek az eljárásból való kilépéskor, ezért az ilyen zártság nem minden esetben megoldás.

Objetum orientált programozás:

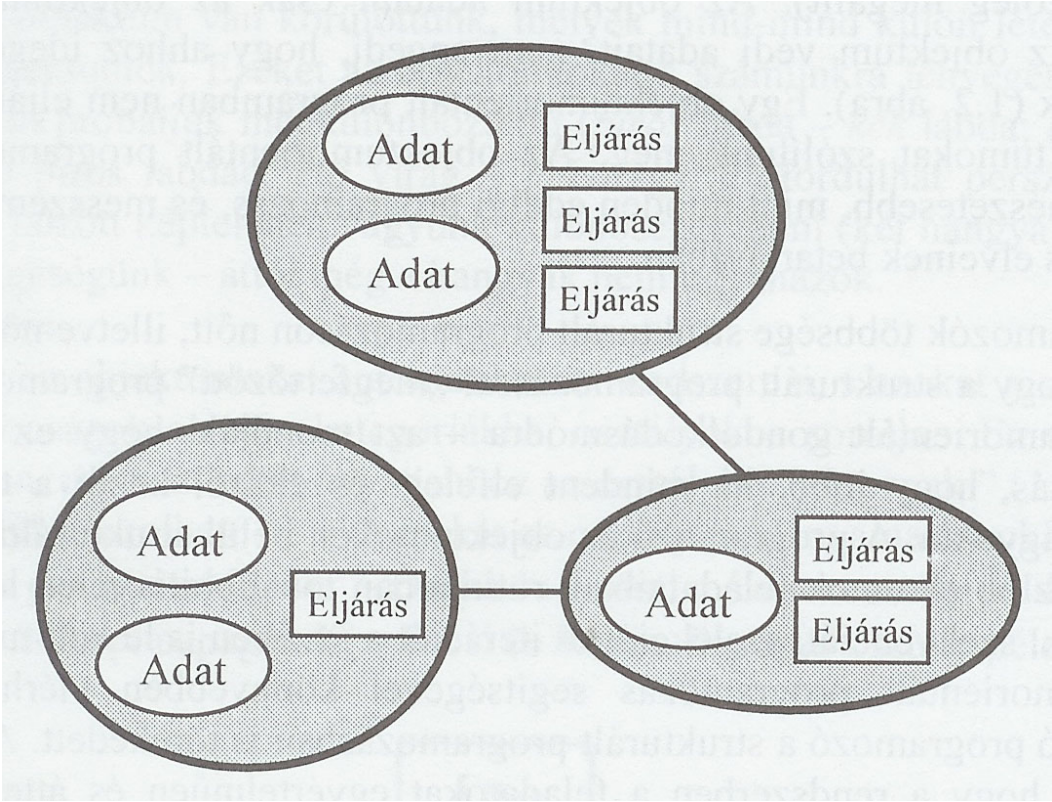
Alan Kay diplomamunkája 1969

Tervek és elképzelések

SmallTalk programozási nyelv megtervezése

Meglévő eljárásorientált nyelveket bővítik OOP lehetőségekkel (Pascal, C)

Új, tisztán OO nyelveket terveznek (Java, C#)



Nyelvek osztályozása:

Hagyományos nyelvek (C)

OOP támogató nyelvek (Pascal, Delphi, C++)

Tisztán OOP nyelvek (C#, Java)

Az OOP alapelvei

Egységbezárás (encapsulation):

az adatok és a hozzájuk tartozó eljárásokat egyetlen egységben kezeljük (objektum-osztály)

Az osztály mezői tárolják az információkat

A metódusok kommunikálnak a külvilággal

Az osztály változóit csak a metódusokon keresztül változtathatjuk meg

A valós világ szemléletesebb leírása

Egységbezárás (encapsulation):

A feladatok elvégzésének „hogyan”-ja az objektum belügye. Az objektum belseje sérthetetlen. Az objektummal csak az interfészen keresztül lehet kommunikálni.

```
type TVerem = class
        VM : integer;
         T : array [1..100] of integer;
        procedure Berak(X:integer);
        function  Kivesz:integer;
        function  Ures:boolean;
        function  Tele:boolean;
        procedure Init;
end;
```

Öröklés (inheritance) :

az objektum-osztályok továbbfejlesztésének lehetősége. Ennek során a származtatott osztály örökli ősétől azok attribútumait, és metódusait, de ezeket bizonyos szabályok mellett újakkal egészítheti ki, és meg is változtathatja

Az eredeti osztályt ősosztálynak nevezzük (szülő)

Az új, továbbfejlesztett osztályt származtatott osztálynak (gyerek)

Egy ősből több leszármaztatott osztályt is készíthetünk

Egy származtatott osztálynak

Öröklés (inheritance) :

Legfeljebb egy szülője lehet (pl.: Pascal, Java, C#) -- öröklődési fa

Több szülője is lehet (pl.: C++) -- öröklődési gráf

Metódusok törzsét megváltoztathatjuk

Mezők neveit, típusait általában nem változtathatjuk meg

Új mezőkkel, és metódusokkal egészíthetjük ki az osztályt

```
type TFejlettVerem = class(TVerem)
        ...
end;
```

Sokalakúság (polymorphism) :

Ugyanarra a kérelemre a különböző objektumok különbözőképpen reagálnak.

a származtatás során az ős osztályok metódusai képesek legyenek az új átdefiniált metódusok használatára újraírás nélkül is

Ezt virtuális metódusokon keresztül érhetjük el

Egyes nyelvekben minden metódus virtuális (pl.: Java)

A metódusoknál külön jelezni kell, melyik a virtuális (pl.: Delphi, C#, C++)

Sokalakúság (polymorphism) :

```
type TElso = object
       ...
       procedure B; virtual;
end;

type TMasodik = class(TElso)
       ...
       procedure B; virtual;
end;
```

Alapfogalmak

Osztály:

Egy felhasználó által készített típus, mely összetett adatszerkezet - elvileg tartalmazza az adott objektum adatait, és az azokat kezelő eljárásokat.

Az objektum információt tárol, és kérésre feladatokat hajt végre.

Az objektum felelős feladatainak korrekt elvégzéséért.

Objektum:

Egy változó, melynek típusa valamely objektumosztály, vagyis az osztály egy példánya.

Objektumorientált program

Egy OO program egymással kommunikáló objektumok összessége, melyben minden objektumnak megvan a jól meghatározott feladata.

Üzenet/kérelem

Az objektumokat üzeneteken keresztül kérjük meg különböző feladatok elvégzésére.

Felelősség

Minden objektumnak megvan a jól meghatározott feladata. Az objektum felelős feladatai elvégzéséért.

Attribútum:

Az osztály egy mezője, konkrét adattárolási képességű adattagja.

Metódus:

Olyan eljárás, mely része valamely objektumosztálynak, így az adott osztály attribútumaival végez valamilyen műveletet.

Példányosítás:

Egy objektumosztályból konkrét objektum készítése (objektum-változó deklarálása)

Inicializálás:

az objektum attribútumainak kezdőérték beállítása, általánosabban az objektum-példány alaphelyzetbe állítása

Objektum-osztály definiálása: (adatmezők + metódusok)

```
type THallgato = class
      Neve: string;
      Eletkora: integer;
      procedure Beiratkozik(Ev,Felev:word);
      ...
end;
```

```
type THallgato = object
      Neve: string;
      Eletkora: integer;
      procedure Beiratkozik(Ev,Felev:word);
      ...
end;
```

Objektum-osztály definiálása: (metódusok kidolgozása)

```
procedure THallgato.Beiratkozik; {param?}
var Neptun:THalozatiKapcsolat;
begin
 Neptun.Connect;
 Neptun.Login('hernyak','jelszo');
 Neptun.SendCommand('beiratk',Ev, Felev);
 Neptun.Close;
end;
```

Példányosítás (object):

```
Var Pisti:THallgato;
BEGIN
   Pisti.Nev := ’Gonosz Pistike’;
   Pisti.Beiratkozik(2004,1);
    …
END.
```

Példányosítás (class):

```
Var Pisti:THallgato;
BEGIN
   Pisti := THallgato.Create;    // e nélkül hiba lesz !
   Pisti.Nev := ’Gonosz Pistike’;
   Pisti.Beiratkozik(2004,1);
    …
END.
```

Távolságok:

Belvilág: Az objektum belvilága -- a saját metódusai.

Külvilág (minden más)

- **Közeli külvilág**: az osztály leszármazottjai, s azok metódusai
- **Távoli külvilág**: a példányokat készítő programmodulok (pl. főprogram)

Adatrejtés (egységbezárás alproblémája) (attributum):

- **PUBLIC**: olyan attribútumok, melyek jellegüknél fogva nem igényelnek speciális szabályozást, azok megváltoztatása nem okoz, nem okozhat problémát az objektum működésében.
- **PRIVATE**: a külvilág nem férhet ezen attribútumukhoz hozzá. Ezek általában segédváltozók, segédmezők.
- **PROTECTED**: olyan attribútumok, melyekhez a távoli külvilág nem férhet hozzá (számára private), de a közeli külvilág, leszármazott osztályok metódusai hozzáférhetnek (számára public)

Adatrejtés (metódusok):

A metódusokhoz való hozzáférést is a fentiek szerint osztályozzuk, de itt az adott metódus meghívhatóságát jelöli:

- PUBLIC: ezen metódusokat a távoli külvilág meghívhatja
- PRIVATE: ezen metódusokat a külvilág nem hívhatja meg, csak az adott osztály metódusai hívhatják meg
- PROTECTED: olyan metódusok, melyeket a távoli külvilág nem hívhat meg (számára private), de a közeli külvilág, a leszármazott osztályok metódusaiból meghívhatóak (számukra public)

```
type THallgato = class

    private
       Neve: string;
       Eletkora: integer;

    public
       procedure Beiratkozik(Ev,Felev:word);
       ...
end;
```

```
Var Pisti:THallgato;
BEGIN
   Pisti.Nev := 'Gonosz Pistike';   // nem működik !
END.
```

Adatrejtés előnyei:

- a mezőkhöz a hozzáférés szabályozható
  - csak azok számára engedjük a közvetlen hozzáférést (írás/olvasás), akikben megbízunk
  - a ”felelősség” miatt szükséges a szabályozás

Adatrejtés hátrányai:

- nem szabályozható, hogy csak az írás, vagy csak az olvasást tiltjuk/engedélyezzük. Egy időben lehet mindkét elérést tiltani/engedélyezni.

Adatrejtés hátrányai (nem elrejtés hátránya):

```
type THallgato = class
    public
       Eletkora: integer;
       ...
end;

...
BEGIN
  Pisti.Eletkora := -489;  // hibás érték !!
  ...
END.
```

Adatrejtés hátrányai (elrejtés hátránya):

```
type THallgato = class
    private
       Eletkora: integer;
       ...
end;


...
BEGIN
  // Pisti.Eletkora := -489;  // nem megy!!
  Writeln(Pisti.Eletkora);  // ez sem megy!!
  ...
END.
```

Adatrejtés megoldása (Set és Get metódus):

```
type THallgato = class
    private
       Eletkora: integer;
    public:
       procedure SetEletkora(ek:integer);
       ...
end;
procedure THallgato.SetEletkora;
begin
  if (ek>0) and (ek<100) Eletkora:=ek
  else ... // hiba jelzése !!!
end;

Pisti.SetEletkora(-486);  // hibajelzés
Pisti.SetEletkora( 18 );  // végrehajtva
```

Adatrejtés megoldása (Set és Get metódus):

```
type THallgato = class
    private:
       Eletkora: integer;
    public:
       function GetEletkora: integer;
       ...
end;
function THallgato.GetEletkora;
begin
  result := Eletkora;
end;

 Writeln( Pisti.Eletkora ); // nem megy
 Writeln( Pisti.getEletkora ); // ok
```

Adatrejtés megoldása (property = virtuális mező):

```
type THallgato = class
    private:
       fEletkora: integer;
       procedure SetEletkora(ek:integer);
    public:
       property Eletkora:integer
           read fEletkora
           write SetEletkora;
       ...
end;
```

Pisti.Eletkora:=126;	↔	**Pisti.SetEletkora(126);**
Writeln(Pisti.Eletkora);	↔	**Writeln(Pisti.fEletkora);**

Read: utána meg kell adni, ha a virtuális mezőt olvassák – mi legyen a teendő:

- **’read’ után lehet írni egy ugyanolyan típusú fizikai mező nevét**
- **’read’ után lehet írni egy paraméter nélküli, megegyező típusú függvény nevét is**

Write: után kell megadni, mi legyen a teendő, ha a virtuális mezőbe új értéket akarnak berakni.

- **’write’ után lehet írni egy ugyanolyan típusú, fizikai mező nevét**

- **’write után lehet írni egy olyan eljárás nevét (ez általában private), amelynek egyetlen paramétere van, amelynek a neve tetszőleges lehet, de a típusa meg kell egyezzen a virtuális mező típusával**

Property: nem kötelező megadni mindkét (read,write) módot. Ha csak a ’read’-t adjuk meg -> csak olvasható virtuális mezőt kapunk. Ha csak a write’-t adjuk meg -> csak írható virtuális mezőt kapunk.

Nem kötelező, hogy a virtuális mező mögött fizikai mező is álljon.

Constructor:

Speciális feladatú metódus: az objektum-példány alaphelyzetbe állítását végzi el (ez általában a mezők kezdőérték-beállítását jelenti).

A ’CLASS’ típusú osztályok példányosításakor kötelező meghívni a konstruktort! Ez a szabály biztosítja, hogy csak már inicializált példánnyal lehessen dolgozni!

```
type THallgato = class
    private:
       fEletkora: integer;
    public:
       constructor Create;
end;

constructor THallgato.Create;
begin
  fEletkora := 0;
  ...
end;
```

Constructor névadása:

A constructor eljárás neve általában ’Create’ vagy ’Init’.

Tetszőleges név választható a constructor-nak.

Több constuctor is lehetséges osztályonként. Jellemző, hogy ezeknek más-más a paramétere (egyébként nincs értelme több konstruktort készíteni).

```
type THallgato = class
       ...
       constructor Create;
       constructor Init(Kezdo:integer);
end;

var H1,H2:THallgato;
begin
 H1 := THallgato.Create;
 H2 := THallgato.Init(24);
 ...
end;
```

Destructor:

Speciális feladatú metódus: az objektum-példány megszűnését megelőzően kerül meghívásra. Ekkor még (utolsó pillanat) a példány által lefoglalt erőforrások (memória, file-ok, hálózati kapcsolatok, …) felszabadítása.

A destructor eljárást meg kell hívni (explicit módon). A meghívás vagy közvetlenül a destruktor metódus hívásával történik. A másik mód, hogy a ’Free’ metódust hívjuk meg, amely meghívja az ’Destroy’ nevű, paraméter nélküli destruktort.

```
type THallgato = class
       destructor Destroy;
       destructor Done(param:integer);
end;

destructor THallgato.Destroy;
begin
 ...
end;

var H1:THallgato;
begin
 H1 := THallgato.Create;
  ...
  H1.Destroy;      //  H1.Free;
end;
```

Öröklődés:

- közös ős: TObject
  - Create constructor
  - Destory virtuális desktrukto
  - Néhány hasznos metódus (Free, InstanceSize, …)

- mezők öröklődése
  - private
  - protected
  - public
- ugyanolyan nevű mező bevezetése
  - private (ok)
  - protected (aut. elfedi)
  - public (aut. elfedi)
- új mező bevezetése

Öröklődés:

- metódus öröklődése
  - private
  - protected
  - public
- ugyanolyan nevű metódus bevezetése
  - private (ok)
  - protected (aut. elfedi, param változhat, működés!)
  - public (aut. elfedi, param változhat, működés!)
- új metódus bevezetése

Öröklődés:

- **virtuális metódusok**
  - **virtual / dynamic**
  - **override**
  - **típus és param lista nem változhat !**
  - **működés !**

VMT tábla
DMT tábla
Konstruktor felelőssége
Konstruktor virtualitása (nincs)
Destruktor virtualitása (van!)