Írta: SAPTippek.hu
2008. December 15. hétfő, 12:58

Az OOP (Objektum Orientált Programozás) alapelvei:

1. Egységbe zárás: az adat és a rajta értelmezett funkcionalitás egy megbonthatatlan egységet képez. (Az attribútumok lokálisak az objektumra nézve.) Az objektumot minden pillanatban az attribútumai által meghatározott állapot jellemzi.
2. Többalakúság (polimorfizmus): az objektum környezet- és állapot érzékenyen képes reagálni.
3. Öröklés: az objektumok származtathatók. Az utód örökli az elől minden öröklésre megengedett attribútumát, és metódusát.

Az objektumok kommunikációja üzenetekkel történik. Az üzenet átadása úgy történik, hogy meghívjuk az objektum valamely metódusát és pl. paraméterként átadjuk neki az üzenetet.
Az objektumokat osztályokkal tipizáljuk. Ez a gyakorlatban azt jelenti, hogy egy ún. osztály (CLASS) definiálja, hogy egy adott objektum példány majdan milyen attribútumokkal (változókkal), illetve milyen funkciókkal (alprogram, metódus) kerül létrehozásra.

Példa:

{xtypo_code} CLASS peldaosztaly DEFINITION.
data: allapot type c length 4.
methods: constructor,
funkcio

IMPORTING import_parameter
type MyType

EXPORTING export_parameter
type OtherType

EXCEPTIONS hiba_tortent.

ENDCLASS.
{/xtypo_code}

Mint látjuk, az osztálynak van egy definíciós része (az ABAP nyelvben). Ebben csak az osztály adattartalmát és interfészét definiáljuk. Az interfész a külvilággal kapcsolatos kommunikációs

felület, ami a gyakorlatban nem más, mint a meghívható metódusok és paramétereik definíciója. Lennie kell tehát egy implementációs résznek is, amely a metódusok kódját tartalmazza. (Más nyelvekben nem válik ketté az interfész és a funkcionalitás, azaz nincs két része az osztálydefiníciónak.)

{xtypo_code}CLASS peldaosztaly IMPLEMENTATION.

* inicializáció

method constructor.

.

endmethod.

* funkcionalitás megvalósítása

method funkcio.

.

endmethod.

ENDCLASS.{/xtypo_code}

Implementációs rész csak abban az esetben nincs, ha az osztályunk egy ún. minta osztály, azaz egy abstract osztály. Ekkor csak ennek leszármazottainak lesznek implementációs részei, azoknak viszont kötelező. A programban az ilyen osztályt így jelöljük:

{xtypo_code}CLASS pelda DEFINITION ABSTRACT. ... ENDCLASS.{/xtypo_code}

Az osztály, illetve metódusai csak a kapott paraméterekkel, illetve az osztály attribútumaival dolgozhatnak. Az egységbe zárás elve szerint az osztály metódusai nem dolgozhatnak (és ne is dolgozzanak!) a program globális változóival, sem a program nem érheti el (néhány publikus jellemző kivételével) az osztály adattagjait.
Hoppá, mit jelent az, hogy publikus jellemző? És milyen a nem publikus?

Példa:

{xtypo_code}CLASS pelda DEFINITION.

PUBLIC SECTION.

data: public_attrib.

methods: public_method.

PROTECTED SECTION.

data: protected_attrib.

methods: protected_method.

PRIVATE SECTION.

data: private_attrib.

methods: private_method.

ENDCLASS.{/xtypo_code}

A publikus (public) részben definiált attribútumok és metódusok elérhetőek a külvilág számára, az interfész szerepét töltik be. Ha nem definiáljuk külön, akkor minden adattag és metódus publikus lesz. Öröklődéskor ezeket a részeket örökli az utód.

A védett (protected) rész elemei csak az aktuális osztály metódusaival érhetők el, illetve a leszármazott osztály (subclass) is örökli ezeket. A saját (private) rész elemei nem öröklődnek és nem látszanak az osztályon kívülről, azt csak az osztály metódusai érik el.

{xtypo_info} **Figyelem!** Mind a privát, mind a védett attribútumok elérhetők az objektumon kívülről, ha azok címét egy metódussal le tudjuk kérdezni. Ez azonban az egységbezárás elvének súlyos megsértése, és nehezen felfedezhető hibák forrása lehet. Kerüljük![/xtypo_info}

A friend osztályok és interfészek (olyan osztályok, melyeknek megengedett a "belátás az osztályunk belsejébe) mind a publikus, mind a védett, mind pedig a saját jelölésű adattagokhoz, illetve metódusokhoz hozzáférnek.
Ezerszer említettük már az öröklést, vagy más néven a leszármaztatást. Nézzünk egy példát:

{xtypo_code}CLASS c1 DEFINITION.

data: s1 type string.

methods: constructor

IMPORTING p_str type string,

print.

ENDCLASS.

CLASS c1 IMPLEMENTATION.

metod constructor.

s1 = p_str.

endmethod.

method print.

write s1.

endmethod.

ENDCLASS.{/xtypo_code}

{xtypo_code}CLASS c2 DEFINITION INHERITING FROM c1.

data: s2 type string.

methods: constructor

IMPORTING p_str type string,

print REDEFINITION.

ENDCLASS.

CLASS c2 IMPLEMENTATION.

metod constructor.

call method super->constructor

exporting p_str = p_str.

s2 = s1.

translate s2 upper case.

endmethod.

method print.

write: s1, s2.

endmethod.

ENDCLASS.{/xtypo_code}

Amiről származtatunk, azt így is nevezik: superclass. A leszármazottat pedig így: subclass. A fenti példában a c1 minden adattagját és metódusát örökli a c2 osztály. Mivel azonban annak is van egy print metódusa, ezért az eredetit felül kellett definiálni.

Meglehetősen sok jellemzője van még az OOP-nak, de most megpróbálunk csak a legfontosabbakra szorítkozni.
Ilyen a létrehozhatóság szabályozása. Az osztályok ugyanis minták (típusok) az objektumokhoz, melyek az osztályok definíciói szerint létrehozott program példányok. Egy osztály a legegyszerűbben így példányosítható:

{xtypo_code}*ABAP objektum

DATA: go_o1 type ref to c1.

CREATE OBJECT go_o1.

*Adatobjektum

TYPES: t_itab type standard table of i.

DATA: gdo_itab type ref to t_itab.

CREATE DATA gdo_itab.{/xtypo_code}

A létrehozhatóság szabályozásával elérhetjük, hogy ne akárki és ne akárhogyan hozhassa létre a kívánt objektumpéldányt.

{xtypo_code}CLASS pelda DEFINITION CREATE PUBLIC.

...

ENDCLASS.{/xtypo_code}

A PUBLIC kulcsszó helyett alkalmazható még a PROTECTED és a PRIVATE is.

PUBLIC: bárki létrehozhat objektumpéldányt az osztályról.

PROTECTED: csak az osztály, illetve leszármazottai metódusai példányosíthatják az objektumot (ekkor
nyilván legalább egy ősnek statikus osztálynak kell lennie - lásd később).

PRIVATE: csak az osztály metódusai példányosíthatják az objektumot (ekkor nyilván legalább egy ősnek
statikus osztálynak kell lennie - lásd később.

Statikus metódusok, statikus attribútumok

Néha előfordul, hogy egy attribútumot az osztály minden objektumpéldányának el kell érnie, méghozzá mindnek ugyanazt az egy fizikai megvalósulást. Ilyen eset például az, amikor a létrehozott példányokat számoljuk.
Néha előfordul, hogy a statikus attribútumokat már a példányosítás előtt el kell érni, például inicializálás céljából. Ilyenkor jöhet jól egy statikus (instancia független = példány független) metódus, amely akkor is végrehajtható, ha nincs objektum példányunk.

Példák a statikus megvalósításokra:

attribútum: CLASS-DATA: counter type i.

metódus: CLASS-METHODS: create_new_object_instance.