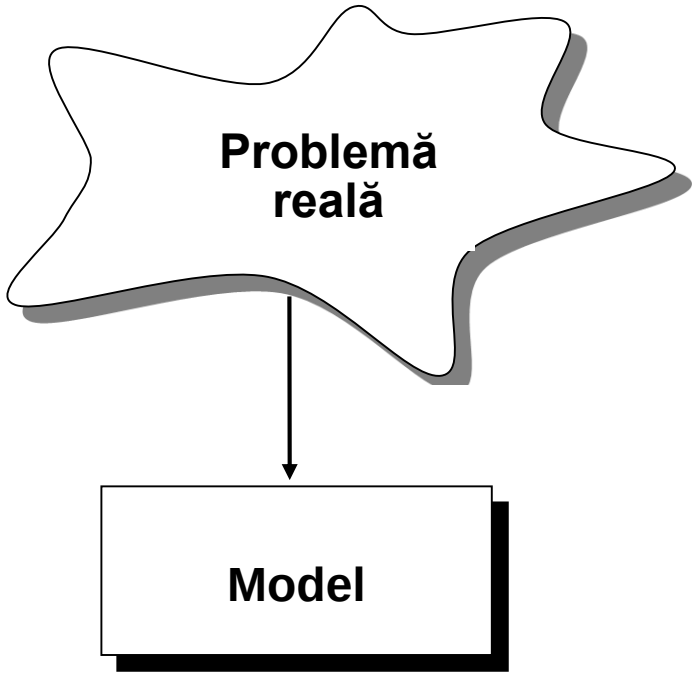
Programare orientată pe obiecte (2)

Cuprins

2 Tipuri abstracte de date

2.1 Tratarea problemelor

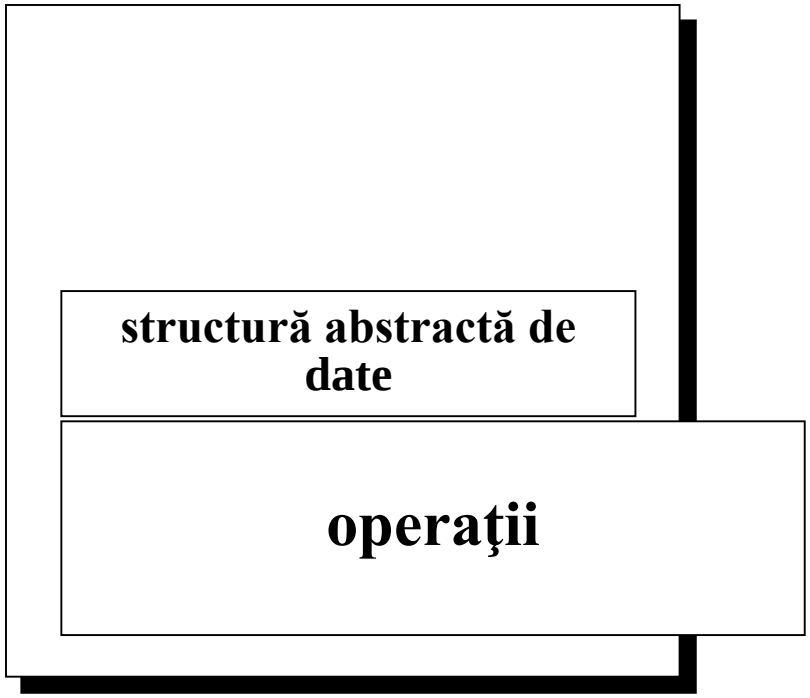
Cel mai frecvent activitatea de programare are ca scop rezolvarea problemelor din "viaţa reală". Aceste probleme apar de obicei cu aspect nebulos şi primul lucru care trebuie făcut este înţelegerea problemei pentru a se separa elementele necesare de cele inutile. Rezultatul acestui efort este o *prezentare abstractă* sau un *model* al problemei.



2.2 Proprietăţi ale tipurilor abstracte de date

Prin abstratizare se crează o entitate bine precizată care să poată fi prelucrată în mod specific. O astfel de entitate defineşte *structura de date* a elementelor unei anumite mulţimi. De exemplu, fiecare *angajat* are datele: *nume, dată de naştere, sex, număr de identificare*.

Structura de date poate fi accesată numai cu *operaţii* definite în mod expres. Acest set de operaţii este numit *interfaţă* şi este *ataşat entităţii*. O entitate cu proprietăţile descrise mai sus este numită *tip abstract de date* (TAD).



interfaţă

După ce se defineşte un TAD, se pot crea *instanţieri* ale sale prin acordarea de valori datelor din structură. De exemplu, atunci când un nou angajat este "creat", structura de date este încărcată cu valorile efective: se obţine astfel o *instanţiere* a unui angajat abstract. Se pot crea atâtea instanţieri ale unui angajat abstract câte sunt necesare pentru a descrie fiecare persoană reală angajată.

2.2 Proprietăţi ale tipurilor abstracte de date (continuare)

Definiţie (Tip abstract de date). *Un* ***tip abstract de date*** *(TAD) este caracterizat de următoarele proprietăţi:*

1. *Are ataşat un un* ***tip****.*
2. *Are ataşat un* ***set de operaţii.*** *Acest set este numit* ***interfaţă****.*
3. *Operaţiile din interfaţă reprezintă* ***unicul*** *mecanism de acces la stuctura de date a tipului.*
4. *Domeniul de aplicabilitate (termen prin care se desemnează atât* ***domeniul valorilor*** *tipului, cât şi* ***domeniile de definiţie ale operaţiilor*** *şi* ***rezultatele operaţiilor****) este definit prin axiome şi precondiţii.*

2.2 Proprietăţi ale tipurilor abstracte de date (continuare)

Exemplu: **definirea TAD *Lista***

1. Când utilizăm reprezentanta definim variabila corespunzătoare ca fiind de tipul *Lista.*

2. Interfaţa către instanţierile tipului *Lista* este asigurată de către fişierul de definiţie a interfeţei.

3. Deoarece definiţia interfeţei nu include reprezentarea efectivă a reprezentantei, ea nu poate fi modificată direct.

4. Domeniul de aplicabilitate este definit de semnificaţia (semantica) operaţiilor introduse. Axiomele şi precondiţiile includ enunţuri ca:

"O listă vidă este o listă"

"Fie l=(d1, d2, d3,..., dN) o listă. Atunci l.append(dM) are ca rezultat l=(d1, d2, d3,..., dN, dM)."

"Primul element al unei liste poate fi şters numai dacă lista nu este vidă."

2.2 Proprietăţi ale tipurilor abstracte de date (continuare)

Importanţa incapsulării structurilor de date

Principiul de a ascunde structura de date utilizată şi de a furniza numai o interfaţă bine definită este numit *incapsulare.*

Separarea dintre structurile de date si operaţii şi constrângerea ca structura de date să fie accesată numai printr-o interfaţă bine definită ne permite să alegem structurile de date cele mai potrivite în contextul unei aplicaţii.

2.3 Tipuri abstracte de date generice

- TAD-urile sunt utilizate pentru a se defini un nou tip, din care pot fi create instanţieri. Ca în exemplul cu listele, uneori aceste instanţieri pot opera şi ele cu diferite tipuri de date. De exemplu, putem să ne gândim la o listă de persoane sau la o listă de maşini sau chiar la o **listă de liste**. Definiţia semantică a listei este mereu aceeaşi. Numai tipul datelor elementare se schimbă după tipul datelor cu care lista operează.
- *Această informaţie adiţională ar putea fi specificată ca un* **parametru generic** care este specificat în momentul creării instanţierii. Astfel, o instanţiere a unui **TAD generic** este de fapt o instanţiere a unei variante particulare a acestui TAD. O listă de persoane poate fi deci declarată, presupunându-se că s-a definit în prealabil tipul de date **Persoana** reprezentând persoanele Lista<Persoana> listaDePersoane;
- Parantezele unghiulare cuprind aici tipul de date pentru care trebuie creată o variantă a TAD-ului generic **Lista**. **listaDePersoane** oferă interfaţa ca orice altă listă, dar operează asupra instanţierilor tipului **Persoana**.

2.4 Notaţii

- Deoarece TAD furnizează un mod abstract de descriere a proprietăţilor unei mulţimi de entităţi, utilizarea lor este independentă de orice limbaj de programare particular. Se introduce următoarea notaţie. Fiecare descriere a unui TAD constă din două părţi:
- **Date**
- Această parte descrie structura de date utilizată în TAD într-un mod neformalizat.
- **Operaţii**
- Această parte descrie operaţiile valide pentru acest TAD, deci descrie interfaţa sa. Utilizăm operaţiile speciale **constructor** pentru a descrie acţiunile care urmează să fie executate când o entitate a acestui TAD este creată şi **destructor** pentru a descrie acţiunile care urmează să fie executate când o entitate este distrusă. Pentru fiecare operaţie se dau *argumente*, *precondiţii* şi *postcondiţii.*

-

2.4 Notaţii (continuare)

Exemplu: **Definiţia TAD *Intreg***

TAD *Intreg* este

Date

O secvenţă de cifre precedată opţional de un semn plus sau minus.

Notăm cu *N* acest număr întreg cu semn.

Operaţii

constructor

Crează un nou întreg.

destructor

Distruge un întreg existent.

sum(k)

Crează un nou întreg care este suma lui *N* şi *k*. În consecinţă, *postcondiţia* operaţiei este *rez=N+k*.

dif(k)

Crează un nou întreg, care este diferenţa dintre *N* şi *k*.

Deci postcondiţia acestei operaţii este *rez=N-k.*

atr(k)

Atribuie lui *N* valoarea *k*. Postcondiţia acestei operaţii

este *N=k.*

...

Sfârşit

3 Concepte ale orientării pe obiecte

3.1 Implementarea tipurilor abstracte de date

Reprezentare pe 2 nivele

Nivelul 1 = definirea unui TAD în care se specifică tot ceea ce se face cu o instanţiere a sa prin invocarea operaţiilor definite. La acest nivel sunt utilizate pre/ şi postcondiţiile pentru a se descrie ceea ce se efectuează.

Exemplu:

{Precondiţie: *L=n* unde *n* este un *Intreg* oarecare}

i.atr(L)

{Postcondiţie: *i=n*}

La nivelul TAD formularea condiţiilor se face cu mijloace matematice.

Nivelul 2 = implementare, în care pentru aplicaţie este aleasă o *reprezentare* efectivă, într-un limbaj de programare.

În limbajul C semnul "=" (egal) implementează operaţia *atr( )*. În limbajul Pascal a fost următoarea reprezentare: *i:=L*; În ambele cazuri este implementată operaţia *atr*. "+" a fost ales pentru a implemeta operaţia *sum* în ambele limbaje, etc.

Observaţie: unele limbaje de programare aleg o reprezentare care este aproape identică cu formularea matematică utilizată în pre- şi postcondiţii. Acest fapt face dificilă departajarea între cele două nivele.

3.2 Clase

Definiţie (Clasă). *O clasă este implementarea unui tip abstract de date (TAD). Ea defineşte atributele şi metodele care implementează structurile de date şi operaţiile din TAD, respectiv.*

3.3 Obiecte

Definiţie (Obiect). *Un obiect este o instanţiere a unei clase. El poate fi identificat în mod unic prin numele său şi în fiecare moment are o stare care reprezintă valorile atributelor sale în acel moment.*

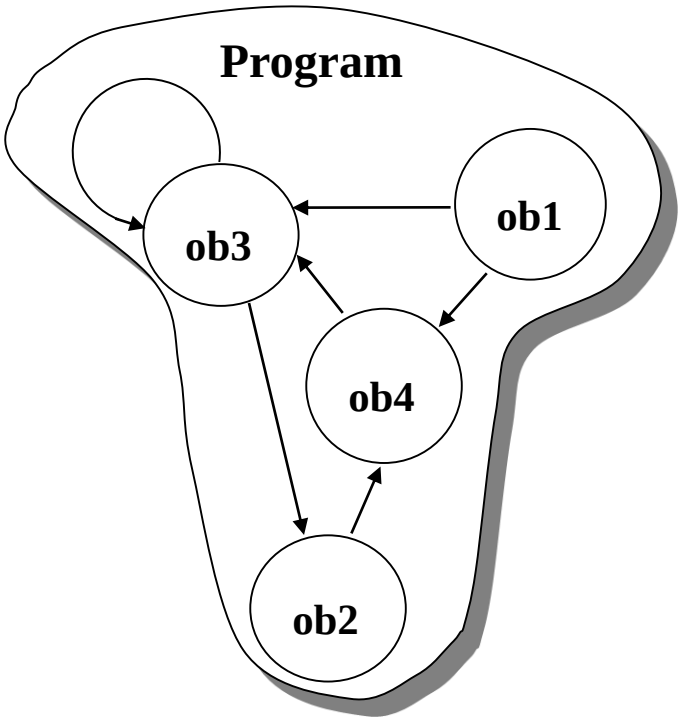
Definiţie (Comportare). *Comportarea unui obiect este definită de mulţimea metodelor care pot fi aplicate asupra sa.*

3.4 Mesaje

Definiţie (Mesaj). ***Un mesaj este o cerere către un obiect de a invoca una din metodele sale.***

Un *mesaj* conţine

- *numele* metodei şi
- *argumentele* metodei.



Considerarea unui program ca o colecţie de obiecte care interacţionează este un principiu fundamental în programarea orientată pe obiecte. Obiectele din această colecţie reacţionează la primirea unor mesaje, schimbându-şi starea în funcţie de invocarea unor metode care poate, la rândul ei, să cauzeze trimiterea altor mesaje către alte obiecte.