9

CONCEPTE DE BAZĂ ALE PROGRAMĂRII ORIENTATE OBIECT

9.1. Introducere
9.2. Abstractizarea datelor
9.3. Moştenirea
9.4. Încapsularea informaţiei
9.5. Legarea dinamică (târzie)
9.6. Alte aspecte

9.1. INTRODUCERE

Termenul "OOP" ("**O**bject **O**riented **P**rogramming") desemnează disciplina programării obiectuale (orientate-obiect). Această disciplină care are la bază ideea unificării datelor cu modalităţile de prelucrare a acestora şi manevrează entităţi reprezentate sub formă de **obiecte** (obiect=date+cod de tratare a acestor date).

Aşa cum s-a subliniat în capitolul 1.3., rezolvarea unei probleme se poate face pe 3 direcţii:
- Rezolvarea *orientată pe algoritm* (pe acţiune), în care organizarea datelor este neesenţială;
- Rezolvarea *orientată pe date*, acţiunile fiind determinate doar de organizarea datelor;
- Rezolvarea *orientată obiect*, care combină tendinţele primelor două abordări.

Programarea obiectuală oferă posibilităţi de **modelare** a obiectelor, a proprietăţilor şi a relaţiilor dintre ele, dar şi posibilitatea de a **descompune** o problemă în componentele sale (soft mai mentenabil, adaptabil, reciclabil). Câteva exemple de limbaje de programare orientată obiect: SIMULA(1965), SIMULA-2(1967), Smalltalk, C++, Java (în plus, Java poate fi considerat un limbaj de programare orientată eveniment).

Facilităţile oferite de programarea orientată obiect (conform lui Pascou) sunt:
1. abstractizarea datelor;
2. moştenirea;
3. încapsularea (ascunderea) informaţiei;
4. legarea dinamică ("târzie").

9.2. ABSTRACTIZAREA DATELOR

Obiectele sunt componente software care modelează fenomene din lumea reală. În general, un fenomen implică tipuri diferite de obiecte. Obiectele care reprezintă aceeaşi idee sau concept sunt de acelaşi **tip** şi pot fi grupate în **clase** (*concrete* sau *abstracte*). Clasele implementează tipuri de date (aşa cum s-a subliniat în capitolul 2, un tip de date înseamnă o mulţime de valori pentru care s-a adoptatat un anumit mod de reprezentare şi o muţime de operatori care pot fi aplicaţi acestor valori), deci şi operatorii destinaţi manipulării acestora: **`Clasă = Date + Operaţii.`**
De exemplu, programatorul îşi poate defini tipul (clasa) `matrice` şi operatorii care pot fi aplicaţi matricilor (* pentru înmulţirea a două matrici, + pentru adunarea a două matrici, - pentru scăderea a două matrici, etc). Astfel, el poate folosi tipul matrice în mod similar unui tip predefinit:

```
matrice A, B;
matrice C=A+B;
```

Tipul unui obiect (şablon al obiectului) este o **clasă**. O clasă se caracterizează prin: numele clasei, atribute, funcţii şi relaţii cu alte clase.
Instanţa este un obiect dintr-o clasă (`A`, `B`, `C` sunt obiecte, instanţe ale clasei `matrice`) şi are proprietăţile definite de clasă. Pentru o clasă definită, se pot crea mai multe instanţe ale acesteia. Toate obiectele au o **stare** şi un **comportament**. ***Starea*** unui obiect se referă la elementele de date conţinute în obiect şi la valorile asociate acestora (datele membre). ***Comportamentul*** unui obiect este determinat de care acţiunile pe care obiectul poate să le execute (metodele).

Atributele specificate în definiția unei clase **descriu valoric proprietățile** obiectelor din clasă, sub diferite aspecte. Cele mai multe limbaje orientate obiect fac următoarea distincție între atribute:
- atribute ale clasei (au aceeaşi valoare pentru toate instanțele clasei);
- atribute ale instanței (variază de la o instanță la alta, fiecare instanță având propria copie a atributului).

În limbajul C++ atributele se numesc **date membre**. Toate datele membre sunt atribute instanță. Atributele de clasă se pot obține în cazul datelor membre statice (aceeaşi adresă de memorare pentru orice instanță a clasei).

Metode (funcții membre). La definirea unei clase se definesc şi metodele acesteia (numite şi funcții membre). Fiecare obiect are acces la un set de funcții care descriu operațiile care pot fi executate asupra lui. Metodele pot fi folosite de instanțele clasei respective, dar şi de instanțele altor clase (prin mecanismul moştenirii).
Clasa conține atât structurile de date necesare descrierii unui obiect, cât şi metodele care pot fi aplicate obiectului. Astfel, gradul de abstractizare este, mult mai ridicat, iar programele devin mult mai uşor de înțeles, depanat sau întreținut.

La crearea unui obiect, alocarea memoriei se poate fi face *static* sau *dinamic* (cu ajutorul unor funcții membre speciale, numite ***constructori***). Eliberarea memoriei se realizează cu ajutorul unor funcții membre speciale, numite ***destructori***, în momentul încheierii existenței obiectului respectiv.

9.3. MOŞTENIREA

Moştenirea este o caracteristică a limbajelor de programare orientate obiect, care permite refolosirea codului şi extinderea funcționalității claselor existente. Între două clase pot exista multe diferențe, dar şi multe asemănări. Este bine ca informația comună unor clase să fie specificată o singură dată (conceptul de ***clasă/subclasă***, ***superclasă/clasă*** în OOP). Mecanismul moştenirii permite crearea unei ierarhii de clase şi trecerea de la clasele generale la cele particulare. Procesul implică la început definirea clasei de bază care stabileşte calitățile comune ale tuturor obiectelor ce vor deriva din bază (ierarhic superioară)(figura 9.1.).
Prin moştenire, un obiect poate prelua proprietățile obiectelor din clasa de bază.

Clasa `A` reprezintă clasa de bază (este o generalizare) şi conține informațiile comune (disponibile prin moştenire şi subclaselor acesteia).
Clasa `B` reprezintă clasa derivată (este o particularizare, o specializare a clasei A) care extinde funcționalitatea clasei de bază şi conține informațiile specifice.
Să presupunem că A reprezintă clasa mamiferelor (cu proprietățile caracteristice: nasc pui vii, au sânge cald, îşi alăptează puii, etc), iar B reprezintă clasa animalelor domestice. În momentul definirii clasei derivate B, aceasta moşteneşte toate caracteristicile clasei A, rămânând de specificat doar trăsăturile distinctive.
În acest caz, A este *clasă de bază*, iar `B` *clasă derivată* (*subclasă* a clasei A).
Sau: `B` este *clasă*, iar A este o *superclasă* a clasei B.
Moştenirea poate fi: ***unică*** sau ***multiplă***.

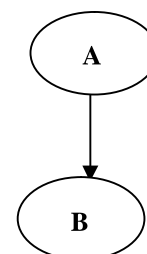

Figura 9.1. Relația clasă de bază-clasă derivată

9.3.1. MOŞTENIREA UNICĂ

În cazul moştenirii unice, fiecare clasă are doar o superclasă. Există *două modalități de specializare* a unei clase de bază*:*
- introducerea de *extra-atribute* şi *extra-metode* în clasa derivată (particulare doar clasei derivate);
- *redefinirea membrilor* în clase derivate *(polimorfism).*

9.3.2. MOŞTENIREA MULTIPLĂ

În situația moştenirii multiple, o clasă are mai multe superclase. Astfel, moştenirea clasei va fi multiplă (rezultând o structură de rețea).

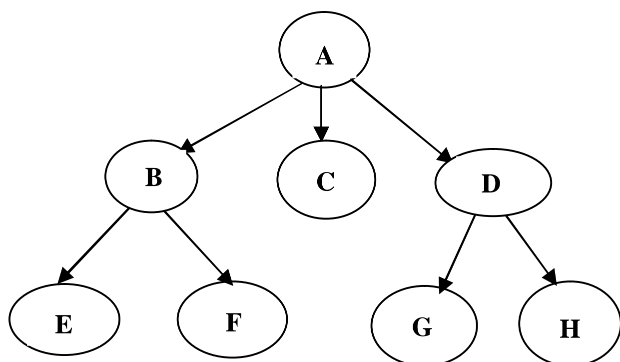

Figura 9.2. Moştenirea simplă (unică)

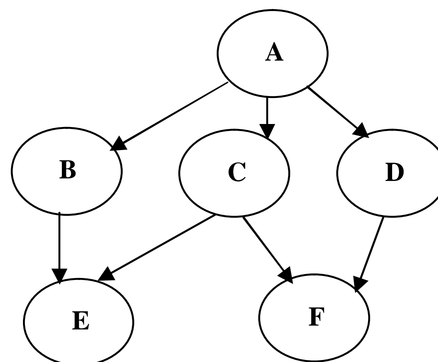

Figura 9.3. Moştenirea multiplă

Moştenirea multiplă este utilă, dar poate crea **ambiguități** (când pentru acelaşi atribut se moştenesc valori diferite). Există mai multe **strategii de rezolvare a conflictului** (părintele cel mai apropiat, cel mai depărtat, etc.). Deasemenea, este posibilă o **moştenire repetată**, în care o clasă ajunge să moştenească de la aceeaşi clasă, pe drumuri diferite în rețea (vezi figura 9.3., în care clasa E moşteneşte de la aceeaşi clasă A, pe drumurile A-B-E, `A-C-E`) . Aşa cum vedea în capitolele următoare, în aceste situații, limbajul C++ oferă programatorului două strategii: 1) clasa E poate avea două copii ale lui A, una pentru fiecare drum; 2) clasa E are o singură copie, iar A este clasă virtuală de bază şi pentru C şi pentru B. Ideea moştenirii multiple poate duce la utilizarea unor **clase pentru care nu există instanțe**, care să ajute doar la organizarea structurii (rețelei) de moştenire. În plus, limbajul C++ permite un control puternic asupra atributelor şi metodelor care vor fi moştenite.

9.4. ÎNCAPSULAREA (ASCUNDEREA) INFORMAȚIEI

Încapsularea (ascunderea) informației reflectă faptul că **atributele instanță şi metodele** unui obiect îl definesc doar pe acesta. Vom spune că metodele şi atributele unui obiect sunt "private", încapsulate în obiect. Interfața cu obiectul relevă foarte puțin din ceea ce se petrece în interiorul lui. Obiectul deține controlul asupra atributelor instanță, care nu pot fi alterate de către alte obiecte. Excepția de la această observație o reprezintă doar *atributele de clasă* care nu sunt încapsulate, fiind partajate între toate instanțele clasei. Această tehnică de "plasare" a valorilor în datele membre private ale obiectului, reprezintă un mecanism de ascundere a datelor.

În limbajul C++ încapsularea poate fi forțată prin controlul accesului, deoarece toate datele şi funcțiile membre sunt caracterizate printr-un **nivel de acces** (rezultând astfel o mare flexibilitate). Nivelul de acces la membrii unei clase poate fi (figura 9.4.):

- `private`: membrii (date şi metode) la care accesul este private pot fi accesați doar prin metodele clasei (nivel acces implicit);
- `protected`: aceşti membri pot fi accesați prin funcțiile membre ale clasei şi funcțiile membre ale clasei derivate;
- `public:` membrii la care accesul este public pot fi accesați din orice punct al domeniului de existență a clasei respective;
- `friend:` aceşti membri pot fi accesați prin funcțiile membre ale funcției prietene specificate.

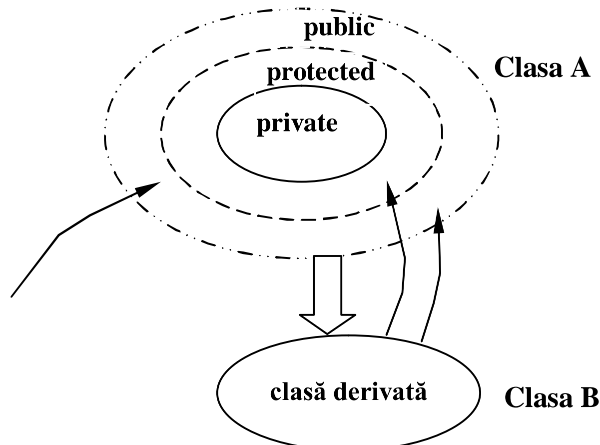

Figura 9.4. Accesul la membrii unei clase

În limbajul C++, nivelul de acces poate preciza şi tipul de moştenire (capitolul 12).

- Publică, unde în clasa derivată nivelul de acces al membrilor este acelaşi ca în clasa de bază;
- Privată, unde membrii protected şi public din clasa bază devin private în clasa derivată.

9.5. LEGAREA DINAMICĂ ("TÂRZIE")

Obiectele unei clase părinte trebuie cunoscute în momentul compilării. Efectul combinat al moştenirii poate determina ca o anumită metodă să fie specializată diferit (prin redefinire), pentru subclase diferite. **Polimorfismul** reprezintă comportamente diferite ale unei metode în raport cu tipul unui obiect. Selectarea unei metode redefinite poate fi realizată în faza de compilare (legarea iniţială), sau în momentul execuţiei (legare târzie). În limbajul C++, legarea dinamică se poate realiza prin implementarea de:
- funcţii virtuale (pot fi redefinite polimorfic);
- funcţii virtuale pure (doar declarate, nu definite).

9.6. ALTE ASPECTE

- **Comunicarea între obiecte**

În limbajele de programare orientate obiect, obiectele comunică între ele prin ***mesaje***, ceea ce conduce la accentuarea conceptului de încapsulare. Un obiect poate "stimula" un altul să activeze (declanşeze) o metodă, trimiţându-i un mesaj. După primirea mesajului, metoda respectivă este apelată cu parametrii furnizaţi, asigurând comportarea corespunzătoare a obiectelor. Metodele sunt invocate prin **trimiterea de mesaje.**
În limbajul C++ **funcţiile membre (metodele) sunt accesate** în mod similar oricarei funcţii, cu deosebirea că este necesară specificarea obiectului căruia îi corespunde metoda.

- **Pseudovariabile**

Limbajele de programare orientate obiect posedă două variabile (numite pseudo-variabile) care diferă de variabilele normale prin faptul că nu li se pot atribui valori în mod direct, de către programator. În general, pseudovariabilele sunt o formă scurtă pentru "***obiectul curent*** " şi pentru ***"clasa părinte a obiectului curent"***. În limbajul C++ există doar una din aceste pseudovariabile, numită **`"this"`** (pointer către obiectul curent).

- **Metaclasele**

Metaclasele reprezintă "clase de clase". O clasă este, de fapt, o instanţă a unei metaclase. *Diferenţele* dintre clase şi metaclase sunt:
  - Clasa defineşte caracteristici (atribute şi metode) ale instanţelor de acel tip. Metodele pot fi folosite doar de obiectele clasei, nu şi de însăşi clasa (restricţie).
  - Metaclasele furnizează un mijloc prin care variabilele clasă pot fi implementate: în unele limbaje OOP, variabilele clasă sunt instanţieri ale unei metaclase.

Limbajul C++ nu include explicit metaclasele, dar suportă *variabilele clasă* sub forma datelor **statice**. Aşa cum funcţiile membre obişnuite sunt încapsulate înăuntrul fiecarei instanţe, pentru o ***funcţie membru statică*** a unei clase, se foloseşte o singură copie, partajată de către toate instanţele clasei. O asemenea funcţie nu este asociată unei anumite instanţe.

- **Persistenţa**

Persistenţa reprezintă *timpul de viaţă al unui obiect* (între crearea obiectului şi ştergerea sa). Instanţele unei clase au un timp de viaţă dat de execuţia unei metode sau a unui bloc, de crearea sau ştergerea specificată explicit în program sau de durata întregului program. Persistenţa obiectelor este importantă în special în aplicaţiile de baze de date.

- **Supraîncarcarea operatorilor**.

Limbajul C++ furnizează modalităţi de *supraîncarcare a operatorilor (overloading)*: acelaşi operator are semnificaţii diferite, care depind de numărul şi tipul argumentelor.