MINISTERUL EDUCAȚIEI, CERCETĂRII ŞI TINERETULUI

Mariana Miloşescu

MODULUL 3

POO şi PV

– Programare orientată pe obiecte şi programare vizuală

Modul opțional

Adresabilitate

- la specializarea *matematică - informatică*
  - Varianta II (BD + POO şi PV)
- la specializarea *matematică - informatică, intensiv informatică*
  - Varianta I (BD + SGBD + POO şi PV)
  - Varianta II (BD + Programare web + POO şi PV)
  - Varianta III (BD + POO şi PV + Programare web)

Manual pentru clasa a XII-a

Filiera teoretică, profil real, specializarea: Matematică - informatică

Filiera vocațională, profil militar, specializarea: Matematică - informatică

(1 oră teorie + 2 ore activități practice) / săptămână, la Varianta I şi Varianta III, Informatică intensiv, şi (3 ore activități practice) / săptămână, la Varianta II, specializarea matematică - informatică, şi la Varianta II de la informatică intensiv

EDITURA DIDACTICĂ ŞI PEDAGOGICĂ, R.A.

Cuprins

3. Programare orientată pe obiecte şi programare vizuală

3.1. Programarea orientată pe obiecte

Competenţe

Generale	Specifice programării orientate pe obiecte
Identificarea datelor care intervin într-o problemă şi aplicarea algoritmilor fundamentali de prelucrare a acestora	
Analizarea unei probleme în scopul identificării şi clasificării datelor necesare	Analizarea problemei care trebuie rezolvată, identificarea sarcinilor aplicaţiei şi a datelor care vor fi prelucrate, şi alegerea claselor adecvate pentru realizarea fiecărei sarcini.
Identificarea relaţiilor dintre date	Identificarea relaţiilor care există între datele aplicaţiei şi, pe baza acestor relaţii, stabilirea grupurilor de date care vor face parte din aceeaşi clasă.
Identificarea modalităţilor adecvate de structurare a datelor care intervin într-o problemă	Identificarea relaţiilor care există între clasele unei aplicaţii şi stabilirea ierarhiei de clase a aplicaţiei.
Utilizarea funcţiilor specifice de prelucrare a datelor structurate	Folosirea metodelor pentru a prelucra datele unei aplicaţii.
Elaborarea algoritmilor de rezolvare a problemelor	
Identificarea tehnicilor de programare adecvate rezolvării unei probleme şi aplicarea creativă a acestora	Identificarea aplicaţiilor care necesită folosirea metodei programării orientate pe obiecte.
Elaborarea strategiei de rezolvare a unei probleme	Structurarea aplicaţiei pornind de la sarcinile pe care trebuie să le rezolve. Identificarea obiectelor care vor rezolva sarcinile aplicaţiei, iar, pentru fiecare obiect, a datelor şi metodelor pe care le va conţine.
Analizarea comparativă a eficienţei diferitelor tehnici de rezolvare a problemei respective şi alegerea celei mai eficiente variante	Analizarea rezolvării unei probleme folosind o aplicaţie realizată prin metoda programării structurate şi o aplicaţie realizată prin metoda programării orientate pe obiecte. Stabilirea metodei celei mai eficiente.
Implementarea algoritmilor într-un limbaj de programare	
Utilizarea instrumentelor de dezvoltare a unei aplicaţii	Utilizarea instrumentelor specifice programării orientate pe obiecte, folosind limbajul C++.
Elaborarea şi realizarea unei aplicaţii, folosind un mediu de programare specific	Construirea unei aplicaţii orientate pe obiecte folosind limbajul C++.
Prezentarea unei aplicaţii	Prezentarea unei aplicaţii orientate pe obiecte pornind de la sarcinile pe care trebuie să le realizeze.

În **programarea clasică** rezolvarea unei probleme se face cu ajutorul algoritmilor descriși prin cele trei structuri de control: structura secvențială, structura alternativă și structura repetitivă. Această metodă de programare este orientată pe prelucrarea datelor, iar programatorul trebuie să construiască un **program** care este format din **ansamblul de date** și **algoritmul** folosit pentru descrierea metodei alese pentru rezolvarea problemei. Algoritmul este implementat în calculator cu ajutorul instrucțiunilor puse la dispoziție de limbajul de programare. Un astfel de limbaj se numește **limbaj procedural** (cum este limbajul **C++**).

Programul este format din instrucțiuni care operează cu date (date elementare sau structuri de date), adică execută acțiuni asupra acestor date. De exemplu, cu structura de date de tip fișier se pot executa operații de scriere în fișier sau de citire din fișier.

Programele de calculator prelucrează informații din lumea care ne înconjoară. Lumea reală este formată din obiecte, iar omul interacționează cu ea prin intermediul acestor obiecte. O fotografie prezintă și ea un ansamblu de obiecte care sunt caracterizate însă numai de proprietăți (formă, dimensiuni, culoare etc.), nu și de acțiuni. Ea este o reprezentare statică a lumii reale. Dar lumea care ne înconjoară este o lume dinamică în care fiecare obiect este determinat de **proprietăți**, **metode** și **evenimente**. Putem să ne gândim la proprietăți ca la atributele obiectului, la metode ca la acțiuni ale obiectului, iar la evenimente ca la răspunsuri ale obiectului la acțiunile omului asupra lui.

Obiectele care diferă între ele numai prin valorile proprietăților și au același comportament (reacționează la aceleași evenimente prin aceleași metode) formează o **clasă de obiecte**.

Studiu de caz

Scop: identificarea proprietăților, metodelor și evenimentelor asociate unui obiect.

Obiectul 1: *Balonul unui copil.*

Proprietățile balonului includ atât atributele vizibile, cum sunt înălțimea, diametrul, culoarea, cele care descriu starea lui (umflat sau dezumflat), cât și atribute invizibile, cum este vârsta. Prin definiție, toate baloanele au aceste proprietăți. Valorile acestor proprietăți diferă de la un balon la altul. Balonului i se pot asocia de asemenea și acțiuni pe care le poate executa, adică metode: metoda de umflare a balonului (acțiunea prin care se umflă balonul cu aer), metoda de dezumflare a balonului (acțiunea prin care se scoate aerul din balon), metoda de înălțare a balonului (acțiunea prin care se ridică balonul). Toate baloanele sunt capabile să execute aceste acțiuni, deci au asociate aceste metode. Baloanele au și răspunsuri predefinite la anumite evenimente externe (la evenimentul înțepătură, răspunsul este dezumflarea automată, iar la evenimentul eliberarea balonului răspunsul este înălțarea lui în aer) sau interne (la evenimentul presiune prea mare a aerului din interior, răspunsul este distrugerea balonului).

Așadar, baloanele diferă între ele numai prin valorile proprietăților, pot să execute aceleași acțiuni (cunosc aceleași metode) și au același comportament (reacționează la aceleași evenimente prin aceleași metode). Putem spune că balonul este un tip de obiect din lumea reală, iar mulțimea baloanelor formează o **clasă de obiecte**.

Obiectul 2: *Câinele.*

Proprietățile câinelui includ atât atributele vizibile, cum sunt rasa, înălțimea, greutatea, culoarea, lungimea părului, cât și atribute invizibile, cum sunt numele și vârsta. Prin definiție, toți câinii au aceste proprietăți. Valorile acestor proprietăți diferă de la un câine la altul. Câinelui i se pot asocia de asemenea și acțiuni pe care le poate executa, adică

metode: lătratul, alergatul, mersul, muşcatul, îngropatul mâncării etc. Toți câinii sunt capabili să execute aceste acțiuni, deci au asociate aceste metode. Câinii au şi răspunsuri predefinite la anumite evenimente externe (la evenimentul prezența unui străin, răspunsul este lătratul, la evenimentul asmuțirea lui asupra unui duşman, răspunsul este muşcatul duşmanului) sau interne (la evenimentul sătul, răspunsul este îngroparea mâncării).

Aşadar, câinii diferă între ei numai prin valorile proprietăților, pot să execute aceleaşi acțiuni (cunosc aceleaşi metode) şi au acelaşi comportament (reacționează la aceleaşi evenimente prin aceleaşi metode). Putem spune că un câine este un tip de obiect din lumea reală, iar mulțimea câinilor formează o **clasă de obiecte**.

Obiectul 3: *Fişierul.*

Proprietățile fişierului includ atributele: nume, tipul informației memorate, dimensiunea, protecția prin parolă, data la care a fost creat, data la care a fost modificat ultima dată, tipul de acces la informația memorată. Prin definiție, toate fişierele au aceste proprietăți. Valorile acestor proprietăți diferă de la un fişier la altul. Fişierului i se pot asocia de asemenea şi acțiuni pe care le poate executa, adică metode: crearea, copierea, mutarea, redenumirea, ştergerea. Toate fişierele sunt capabile să execute aceste acțiuni, deci au asociate aceste metode. Fişierele au şi răspunsuri predefinite la anumite evenimente externe: la evenimentul deschiderea fişierului, răspunsul este alocarea unei zone de memorie internă în care sunt transferate înregistrările din fişier care se vor prelucra; la evenimentul închiderea fişierului, răspunsul este eliberarea zonei de memorie alocată fişierului pentru lucru; iar la evenimentul comunicarea parolei, răspunsul este deschiderea fişierului, în cazul în care parola este corectă.

Aşadar, fişierele diferă între ele numai prin valorile proprietăților, pot să execute aceleaşi acțiuni (cunosc aceleaşi metode) şi au acelaşi comportament (reacționează la aceleaşi evenimente prin aceleaşi metode). Putem spune că fişierul este un tip de obiect din lumea reală, iar mulțimea fişierelor formează o **clasă de obiecte**.

Aşa cum o fotografie descrie starea unui obiect la un moment dat, şi o structură de date folosită pentru descrierea obiectului nu poate să caracterizeze decât starea acelui obiect la un moment dat, deci este o descriere statică a obiectului. Tendința limbajelor moderne de programare este de a pune în corespondență obiectele reale cu obiectele virtuale – pentru a putea evidenția transformările pe care acestea le suferă în mod continuu. Obiectul va încapsula atât datele care descriu proprietățile, cât şi subprogramele care prelucrează aceste date şi care vor defini metodele obiectului.

Această metodă de programare se numeşte **programarea orientată pe obiecte** – deoarece rezolvarea problemei se face cu ajutorul obiectelor. Ea este orientată pe definirea obiectelor. Folosind această metodă, programatorul trebuie să construiască **obiecte** care sunt formate din **date** şi **subprogramele** care prelucrează aceste date.

Aceeaşi problemă poate fi rezolvată cu oricare dintre cele două metode. Metoda programării orientate pe obiecte are însă următoarele avantaje: permite un control mai bun al programelor (în special, al programelor de dimensiuni mari) şi permite o mai uşoară dezvoltare a programelor.

Pentru metoda programării orientate pe obiecte poate fi folosit orice limbaj de programare. Limbajele care sunt **orientate pe programarea pe obiecte** (cum este limbajul **C++**) prezintă avantajul că au implementate structuri de date care permit încapsularea datelor şi a subprogramelor într-o singură variabilă de memorie.

Domeniul în care programarea orientată pe obiecte a ajuns indispensabilă este realizarea interfeței. Aşa cum obiectele cu care venim în contact reprezintă interfața între noi şi lumea reală, şi interfața grafică dintre utilizator şi programul de aplicație este construită cu diferite obiecte virtuale, afişate pe ecran, care pot fi acționate cu mouse-ul şi tastatura. Putem spune că, în urma acționării, obiectul declanşează un anumit eveniment. Deoarece utilizatorul poate veni în contact cu mai multe aplicații, s-a căutat o standardizare a acestor obiecte, atât din punct de vedere al proprietăților, cât şi din punct de vedere al răspunsului la evenimentul declanşat. De exemplu, dacă un utilizator vede pe ecranul calculatorului un buton pe care scrie *OK*, ştie că dacă îl va acționa se va închide caseta de dialog şi se va continua execuția aplicației.

3.1.1. Principiile programării orientate pe obiecte

Conceptele folosite în programarea orientată pe obiecte sunt următoarele:

Obiectul reprezintă un ansamblu de date (un număr fix de variabile) şi subprograme necesare pentru prelucrarea datelor.

Datele poartă numele de **proprietăți**, iar subprogramele – numele de **metode** sau **evenimente**. Proprietățile, metodele şi evenimentele sunt **membrii** unui obiect. Interfața de acces la obiect este realizată numai prin intermediul metodelor. În acest mod, subprogramele membre ale unui obiect prelucrează datele membre ale obiectului care apelează metoda.

Proprietatea reprezintă un atribut al unui obiect care defineşte una dintre caracteristicile sau unul dintre aspectele sale. De exemplu, un obiect vizual al interfeței poate avea proprietatea *visible* – care determină dacă obiectul este vizibil la un moment dat.

Metoda reprezintă acțiunea pe care o poate executa un obiect. Utilizatorul unui obiect are acces la datele obiectului numai prin intermediul metodelor obiectului, iar metoda are acces implicit la membrii unui obiect. Metodele ascund celui care utilizează obiectul amănunte despre modul în care a fost implementat. De exemplu, un obiect vizual de tip *listă* are implementate metode necesare pentru întreținerea listei: adăugarea unui articol la listă, ştergerea unui articol din listă etc.

Evenimentul reprezintă o acțiune recunoscută de obiect pentru care se poate scrie un program ca răspuns. Evenimentele pot fi *externe*, adică generate de acțiuni ale utilizatorului (un clic cu mouse-ul, mişcarea mouse-ului, apăsarea unei taste) sau *interne*, adică generate printr-un cod de program sau de sistem.

Clasa reprezintă definiția unui anumit tip de obiect. Definiția cuprinde descrierea proprietăților şi a metodelor obiectului.

Folosirea claselor permite gestionarea mai multor obiecte de acelaşi tip. Clasa este însă doar un termen abstract, un şablon care defineşte caracteristicile unui obiect (cum arată şi cum se comportă).

Instanța reprezintă un obiect creat pornind de la definiția unei clase.

Spre deosebire de clasă, care este doar o definiție a obiectului, o instanță există ca un obiect care poate fi folosit pentru a executa anumite acțiuni. De exemplu, o casetă de text dintr-un formular este o instanță a clasei *TextBox* care descrie acest tip de obiecte vizuale, fişerul *alfa.txt* este o instanță a clasei *fişier*, iar câinele *Azor* este o instanță a clasei *Câine*.

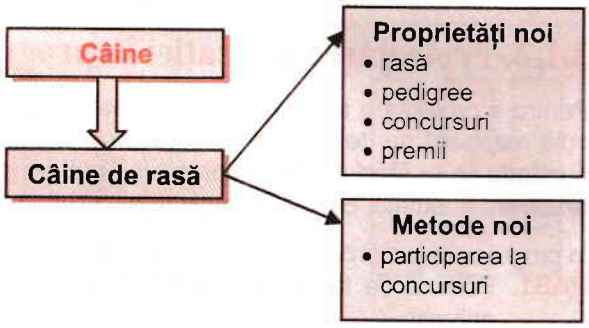
Instanțe multiple reprezintă mai multe obiecte create de aceeaşi clasă. Obiectele au propriile proprietăți şi date private, dar folosesc împreună codul de program.

Încapsularea reprezintă un mecanism prin care, folosind o singură definiție, sunt incluse într-o singură structură de date toate componentele unui obiect: datele şi metodele.

Datele membre ale unui obiect nu pot fi modificate decât prin intermediul metodelor proprii obiectului respectiv. Încapsularea izolează complexitatea internă a unui obiect de restul aplicației.

Moştenirea este o facilitate, oferită de programarea pe obiecte, prin care o clasă nouă, numită clasă derivată, se poate construi pornind de la o clasă existentă, numită clasă de bază, prin preluarea caracteristicilor clasei de bază.

Clasa derivată se mai numeşte şi **clasă copil**, iar clasa de bază – **clasă părinte**. O clasă derivată poate fi folosită pentru a obține din ea o altă clasă derivată, ea devenind clasa de bază pentru noua clasă. Folosind facilitatea de moştenire, se poate crea o **ierarhie de clase** şi se poate dezvolta uşor un software existent. Clasa copil moşteneşte accesul la datele şi metodele strămoşilor (de exemplu, dacă o clasă *Text* are proprietatea ca textul să fie scris îngroşat, orice urmaş al acestei clase va avea această proprietate).



La proprietățile şi metodele clasei de bază, programatorul poate adăuga proprietăți şi metode noi, definite explicit în noua clasă. De exemplu, din clasa de bază *Câine* se poate obține clasa derivată *Câine de rasă*. Clasa *Câine de rasă* va moşteni toate proprietățile şi toate metodele clasei *Câine*, la care se vor adăuga proprietăți şi metode noi.

Orice modificare într-un strămoş se reflectă şi la urmaşi.

Principalele **avantaje ale folosirii programării orientate pe obiecte** sunt:

→ **Reutilizarea unui software deja scris**. Acest avantaj creşte eficiența programatorilor. În programarea clasică, se realiza prin crearea bibliotecilor de subprograme. În programarea orientată pe obiecte se realizează prin crearea bibliotecilor de obiecte. Astfel, un obiect creat pentru o aplicație poate fi folosit şi într-o altă aplicație.

→ **Dezvoltarea mai uşoară a aplicațiilor**. În programarea orientată pe obiecte creşte gradul de modularizare al unei aplicații față de programarea clasică. Adăugarea sau modificarea unor module de aplicație se va face mai uşor datorită organizării obiectelor în clase de obiecte şi a facilității de moştenire.

→ **Controlul mai bun al programelor de dimensiuni mari**. Aplicația este mai bine *structurată* decât în cazul programării clasice, şi mult mai *abstractizată*, permițând o mai uşoară citire şi urmărire a codului.

→ **Dezvoltarea programării vizuale ca aplicație a programării orientate pe obiecte**. În varianta clasică, descrierea unui obiect grafic de pe ecran se făcea precizând printr-un set de instrucțiuni caracteristicile obiectului (coordonatele obiectului, dimensiunile, cu-

loarea, textul afişat etc.), iar prin subprograme – modul în care putea fi manipulat obiectul. În programarea vizuală există deja *obiecte vizuale predefinite* care pot fi folosite pentru construirea unei interfețe. Obiectul vizual conține proprietățile obiectului grafic (cu valori implicite, care însă pot fi modificate) şi metodele şi evenimentele la care poate reacționa obiectul vizual, fără să mai necesite scrierea codului de către programator.

a. Definiți următoarele clase de obiecte: pisică, elev, casă, autoturism, carte. Pentru fiecare clasă de obiecte veți preciza proprietățile, metodele şi evenimentele.

b. Descompuneți obiectul *autoturism* în obiecte distincte, mai simple. Precizați pentru fiecare obiect proprietățile, metodele şi evenimentele.

c. Descompuneți obiectul *şcoală* în obiecte distincte, mai simple. Precizați pentru fiecare obiect proprietățile, metodele şi evenimentele.

d. Pornind de la clasa de obiecte *fişier*, definiți clasele derivate *fişier de date* şi *fişier executabil*. Precizați pentru fiecare clasă derivată metodele noi pe care le conține.

e. Pornind de la clasa de obiecte *fişier de date*, definiți clasele derivate *fişier document* şi *fişier foaie de calcul*. Precizați pentru fiecare clasă derivată metodele noi pe care le conține.

3.1.2. Proiectarea aplicației în programarea orientată pe obiecte

Pentru a proiecta o aplicație, folosind metoda de programare clasică, problema care trebuie rezolvată se descompune în subprobleme (**module**), fiecare subproblemă descompunându-se la rândul ei în subprobleme mai simple, până când se obțin subprobleme cu rezolvare imediată. Subproblemele sunt rezolvate cu ajutorul **subprogramelor**.

În programarea orientată pe obiecte, pentru a proiecta o aplicație, se execută următorii paşi:

PAS1. Pornind de la funcțiile aplicației, se identifică obiectele principale care alcătuiesc aplicația.

PAS2. Obiectele complexe sunt descompuse în obiecte distincte, mai simple.

PAS3. Pentru fiecare obiect se identifică sarcina pe care o are în realizarea aplicației.

PAS4. Pentru fiecare obiect, pornind de la sarcina pe care o are în cadrul aplicației, se identifică operațiile pe care le execută obiectul sau operațiile pe care le execută programul asupra obiectului. Aceste operații vor deveni metodele obiectului.

PAS5. Pentru fiecare obiect, pornind de la metodele pe care trebuie să le cunoască, se identifică informațiile pe care trebuie să le conțină pentru a putea executa metodele. Aceste informații vor deveni proprietățile obiectului.

Studiu de caz

Scop: proiectarea unei aplicații folosind metoda programării orientate pe obiecte.

Enunțul problemei. *Să se proiecteze o aplicație de tip Editor de texte.*

O aplicație de tip editor de texte are următoarele funcții:

→ editarea textului;
→ tipărirea textului;
→ păstrarea textului în fişier.

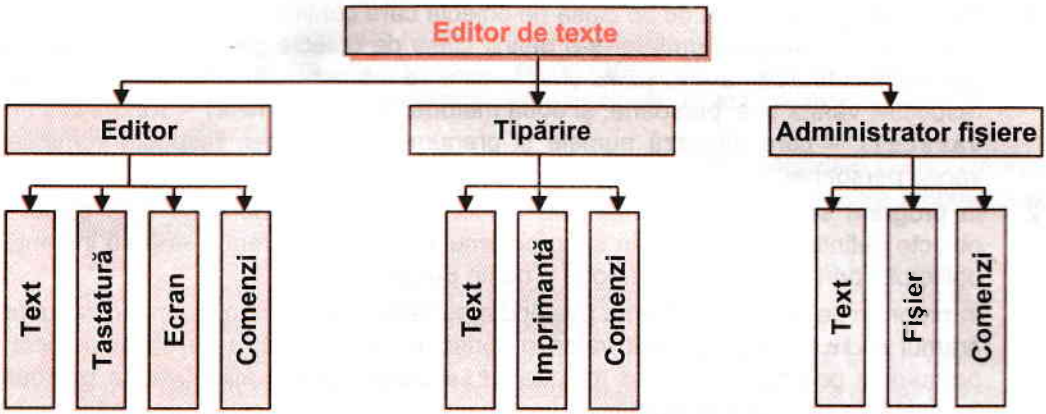
Principalele obiecte ale unei aplicații de tip *Editor de texte* vor fi: *Editorul*, *Tipărirea* şi *Administratorul de fişiere*.

Sarcina editorului este de a asigura scrierea textului de la tastatură, afişarea textului pe ecran, operaţii de corectură în text prin ştergerea şi inserarea caracterelor, operaţii cu blocuri de text (selectarea, copierea, mutarea şi ştergerea blocului de text) şi operaţii de căutare a unui şir de caractere, respectiv de căutare şi înlocuire a unui şir de caractere cu un alt şir de caractere.

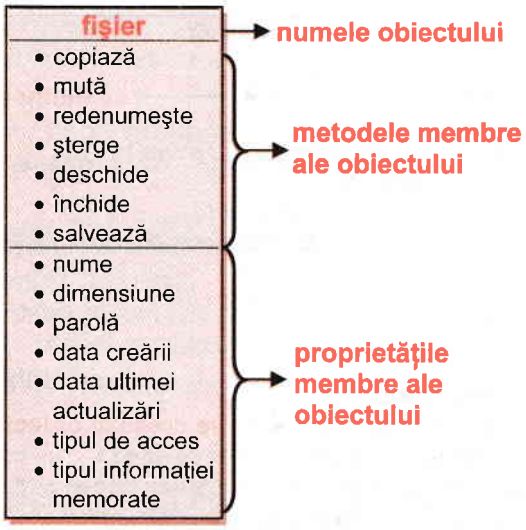
Sarcina tipăririi este de a asigura pregătirea textului pentru tipărire (dimensiunea paginii fizice, dimensiunea paginii utile, orientarea la tipărire, antetul şi subsolul paginii), numărul de copii, ce se tipăreşte din text (tot textul, zona selectată sau paginile precizate), imprimanta la care se tipăreşte şi tipărirea textului.

Sarcina administratorului de fişiere este de a asigura salvarea textului într-un fişier, încărcarea textului din fişier pentru a fi prelucrat şi operaţiile obişnuite cu fişiere (copiere, ştergere, redenumire etc.).

Pornind de la sarcinile definite anterior, fiecare dintre cele trei obiecte poate fi descompus la rândul său în alte obiecte.



După ce au fost identificate toate obiectele aplicaţiei, se trece la identificarea proprietăţilor şi metodelor fiecărui obiect. De exemplu, scopul obiectului **fişier de date** în cadrul unei aplicaţii este de a memora o colecţie de date. Din această cauză, el va trebui să cunoască atât metodele folosite de orice obiect *fişier* cât şi metodele de deschidere, de închidere şi de salvare a informaţiilor. Pentru a putea executa aceste metode, are nevoie de următoarele informaţii: nume, dimensiune, parolă de protecţie, data la care a fost creat, data la care a fost actualizat, tipul de acces.



a. Pentru exemplul precedent, enumeraţi câteva dintre proprietăţile şi metodele obiectului editor şi ale obiectului tipărire.

b. Proiectaţi o aplicaţie pentru exploatarea listelor.

3.1.3. Structura unei aplicaţii orientate pe obiecte

În metoda clasică de programare, programul conţine instrucţiuni pentru definirea datelor şi instrucţiuni care implementează algoritmul folosit pentru rezolvarea problemei. Algoritmul precizează modul în care sunt prelucrate datele pentru a obţine rezultatele dorite.

În metoda programării orientate pe obiecte se folosesc obiecte care încapsulează atât datele, cât şi o serie de subprograme care reprezintă metodele cunoscute de obiect. În programul care lucrează cu aceste obiecte nu mai sunt necesare instrucţiuni pentru a descrie algoritmul de rezolvare a problemei ci, pentru a rezolva problema, se apelează metodele obiectului. Cu alte cuvinte, programul nu mai este o listă de instrucţiuni prin care se precizează ce operaţii trebuie să se execute cu datele definite, ci cereri către obiect de a executa singur anumite prelucrări.

Structura unui program orientat pe obiecte este următoarea:

1. Se defineşte o structură de tip clasă de obiecte care conţine proprietăţile şi metodele obiectului. În exemplul următor s-a definit clasa de obiecte `persoana` care conţine proprietăţile (datele) `nume`, `pren` şi `v`, în care se memorează numele, prenumele şi, respectiv, vârsta unei persoane, şi două metode (subprogramele) – `afiseaza()` şi `varsta()` – care afişează numele şi prenumele persoanei, respectiv furnizează vârsta persoanei.
2. În program se creează instanţe ale obiectului, adică variabile de tipul clasei de obiecte definită. În exemplu, în subprogramul `main()` s-a creat o singură instanţă a obiectului prin variabila de memorie `p` de tip `persoana`.
3. În program se rezolvă problema apelând metodele obiectului. În exemplu, în subprogramul `main()`, s-au atribuit valori proprietăţilor obiectului, s-au afişat numele şi prenumele persoanei apelând metoda `afiseaza()` şi s-a afişat vârsta persoanei furnizată de metoda `varsta()`.

```
#include<iostream.h>

class persoana                    ← se defineşte clasa de obiecte persoana
 {public:
    char nume[20],pren[20];       ← proprietăţile clasei de obiecte
    int v;
    void afiseaza()
      {cout<<nume<<" "<<pren<<endl;}
    int varsta()                  ← metodele clasei de obiecte
      {return v;}
 };

void main()                       se atribuie valori proprietăţilor obiectului p
{persoana p;                      ← se creează obiectul p
 cout<<"Numele persoanei: "; cin.get(p.nume,20); cin.get();
 cout<<"Prenumele persoanei: "; cin.get(p.pren,20);
 cout<<"Varsta: "; cin>>p.v;
 p.afiseaza();                    ← se apelează metodele obiectului p
 cout<<"are "<<p.varsta()<<" ani";
}
```

3.1.4. Clase şi obiecte

3.1.4.1. Definirea claselor şi a obiectelor

Sintaxa instrucțiunii pentru definirea unei clase este următoarea:

```
class nume_clasa
 {//date şi subprograme private
  specificator_de_acces:
    //date şi subprograme
.....................................
  specificator_de_acces:
    //date şi subprograme
 };
```

Definirea unei clase de obiecte se termină cu caracterul ; deoarece este o instrucțiune.

Specificatorii de acces precizează modul în care programul are acces la membrii obiectului. Un specificator de acces are efect începând cu declararea lui şi până la întâlnirea unui alt specificator de acces sau până la sfârşitul definiției clasei. Există următorii specificatori de acces la membrii obiectului:

1. **public** – Programul are acces la aceşti membri ai obiectului. Metodele publice se mai numesc şi **metode de interfață** deoarece ele definesc operațiile pe care programul le poate executa asupra obiectului.
2. **private** – Programul nu are acces direct la aceşti membri ai obiectului. La aceşti membri accesul direct este permis numai din interiorul obiectului. Programul are acces la aceşti membri numai prin intermediul metodelor publice.
3. **protected** – Pentru clasa de bază programul are acces direct la aceşti membri ai obiectului. Pentru o clasă derivată programul nu are acces direct la aceşti membri ai obiectului, accesul fiind permis numai din interiorul obiectului.

Specificatorul de acces implicit este **potected**. Altfel spus, dacă nu este precizat niciun specificator de acces, programul nu are acces la membrii obiectului.

Printr-o instrucțiune de definire a unei clase nu se creează o variabilă de memorie, ci un tip de dată utilizator (tipul de dată clasă). Pentru a folosi în program o variabilă de memorie de acest tip, ea trebuie declarată:

```
nume_clasă   nume_variabilă;
```

Aşadar, pentru a putea manipula un obiect trebuie să definiți:

1. un tip de dată clasă de obiecte prin care descrieți colecția de membri ai obiectului şi
2. o variabilă de memorie care va avea ca tip de dată numele clasei (instanța obiectului).

Se pot folosi următoarele două variante de declarare a tipului clasă de obiecte şi a variabilei de acest tip:

Varianta 1

```
class nume_clasă {//definire membri clasă};
nume_clasă nume_variabilă;
```

Varianta 2

```
class nume_clasă
 {//definire membri clasă} nume_variabilă;
```

Accesul la membrii obiectului se face folosind **operatorul punct (.)**:

```
nume_variabilă_obiect.nume_membru
```

Operatorul punct este **operatorul de selecţie a membrului unui obiect** şi leagă numele obiectului de numele membrului, adică, în expresia **a.b** – *a* trebuie să fie de tip obiect, iar *b* trebuie să fie de tip membru al acelui obiect. Punctul este un **operator binar** şi are **prioritate maximă**.

Observaţie. Definirea metodelor unei clase de obiecte se poate face fie în interiorul clasei, fie în exteriorul ei. În programul din paragraful precedent, membrii clasei `persoana` au fost definiţi în interiorul clasei. În programul următor, membrii clasei sunt definiţi în afara clasei. În acest caz, definiţia clasei va conţine prototipurile funcţiilor membre.

```
#include<iostream.h>

class persoana
 {public:
    char nume[20],pren[20];
    int v;
    void afiseaza();
    int varsta();
  };
void persoana::afiseaza()
      {cout<<nume<<" "<<pren<<endl;}
int persoana::varsta()
      {return v;}

void main()
{persoana p;
 cout<<"Numele persoanei: "; cin.get(p.nume,20); cin.get();
 cout<<"Prenumele persoanei: "; cin.get(p.pren,20);
 cout<<"Varsta: "; cin>>p.v;
 p.afiseaza(); cout<<"are "<<p.varsta()<<" ani";
}
```

metodele sunt declarate în interiorul clasei de obiecte

metodele sunt definite în exteriorul clasei de obiecte

Operatorul **::** se numeşte **operator de rezoluţie a domeniului de vizibilitate** a unui identificator şi leagă numele domeniului de vizibilitate de un identificator (numele unei variabile de memorie sau numele unui subprogram). Acest operator este un **operator binar** şi are **prioritate maximă**. El permite accesul la un identificator declarat într-un domeniu închis. În exemplul precedent operatorul **::** leagă numele clasei (care este un domeniu închis) de numele unui membru al clasei (numele unui subprogram).

În definirea unei clase se pot folosi ambele metode; fiecare metodă are avantaje şi dezavantaje:

1. Dacă funcţia este declarată în interiorul clasei, va fi tratată de compilator ca funcţie `inline`, adică fiecare apel al funcţiei este înlocuit cu codul său. Acest mod de tratare are **avantajul** că se reduce timpul de execuţie a programului prin eliminarea timpilor de încărcare a stivei şi de revenire după execuţie, care apar la fiecare apel al unui subprogram. Are însă **dezavantajul** unui consum mai mare de memorie deoarece creşte codul programului. În cazul în care metoda este definită în interiorul clasei, pentru fiecare instanţă a obiectului care apelează metoda se creează o copie a codului metodei. În cazul în care metoda este definită în afara clasei, instanţele multiple ale obiectelor îşi partajează o singură copie a codului metodei.