Programare Orientata pe Obiect
Lucrarea de laborator Nr. 6

Functii virtuale si polimorfism.

În lucrarea precedenta s-a ilustrat folosirea cuvântului cheie *virtual* al limbajului C++, înaintea numelui clasei de baza la definirea claselor derivate pentru cazul de mostenire multipla, în care o clasa era mostenita indirect de mai multe ori, evitându-se astfel crearea de copii multiple care generau ambiguitati. Cuvântul cheie *virtual* mai este folosit la definirea *functiilor virtuale* care reprezinta un mecanism pentru implementarea *polimorfismului* în clasele derivate. Doua sau mai multe obiecte sunt polimorfe daca au asemanari, dar totusi sunt diferite. Supraîncarcarea operatorilor si a functiilor sunt exemple de polimorfism pentru ca o singura entitate refera doua sau mai multe caracteristici diferite. Altfel spus, polimorfismul permite definirea unei interfete comune pentru mai multe metode specifice diferitelor functionalitati.

1. Conversia pointerilor între clase de baza si derivate. Functii virtuale.

Conversia unui pointer la o clasa derivata în pointer la o clasa de baza a acesteia este implicita, daca derivarea este de tip `public` si nu exista ambiguitati. Rezultatul conversiei este un pointer la subobiectul din clasa de baza al obiectului din clasa derivata.

Conversia inversa, a unui pointer la o clasa de baza în pointer la derivata nu este admisa implicit. O astfel de conversie se poate forta explicit prin operatorul de conversie `cast`. Rezultatul unei astfel de conversii este însa nedeterminat, de cele mai multe ori provocând erori de executie. Exemplificare:

```
class Baza { /*    */ };
class Derivata:public Baza { /*    */ };
void main(){
      Derivata ob_d;
      Baza* p_ob_b = &ob_d;   // corect, conversie implicita
      Baza ob_b;
      Derivata* p_ob_d = &ob_b;// eroare la compilare, nu se poate converti
                              // implicit de la class Baza* la class Derivata*
      Derivata* p_ob_d =(Derivata*)&ob_b;  // se compileaza corect, dar
                                           //rezultatul este nedeterminat
}
```

Exercitiul 1:

Definiti o clasa `Vehicul` care contine o functie membra de afisare a numelui clasei. Din clasa `Vehicul` derivati o clasa `Autoturism` în care redefiniti functia de afisare. În functia `main()`:

- Creati câte doua obiecte instante ale celor doua clase, dintre care câte unul prin alocare dinamica de memorie. Apelati functia de afisare pentru fiecare din cele 4 obiecte. Ce mesaje sunt afisate la rularea programului?
- Declarati un pointer la clasa de baza si folositi-l pentru a crea dinamic un obiect de tipul clasa derivata. Apelati functia de afisare pentru noul obiect. Ce mesaje sunt afisate la rulare?

Modificati programul adaugând înaintea functiei de afisare (înainte de tipul returnat) din clasa de baza adaugati cuvântul `virtual`. Compilati si rulati din nou programul. Ce mesaje sunt afisate? Observati deosebirile între afisari la cele doua rulari ale programului.

Un pointer la o clasa de baza poate fi folosit pentru referentierea unei întregi ierarhii de clase. De exemplu, daca referentiem clasa `Vehicul` ne putem referi la un autoturism (pentru care s-a definit clasa derivata `Autoturism`), camion, motocicleta sau orice alt mijloc de transport. Termenul general de vehicul

poate referentia tipuri diferite, pe când un termen mai precis, de exemplu, autoturismul nu poate indica decât un autoturism. Astfel, în C++, un pointer care referentiaza o clasa de baza va putea fi folosit si pentru referentierea claselor derivate. Un pointer catre un obiect din clasa `Autoturism`, nu va putea fi folosit pentru referentierea clasei `Vehicul` pentru ca este „prea" specific.

O *functie virtuala* este o functie care este declarata de tip `virtual` în clasa de baza si redefinita într-o clasa derivata. Redefinirea unei functii virtuale într-o clasa derivata domina (*override*) definitia functiei în clasa de baza. Mecanismul de virtualitate se manifesta numai în cazul apelului functiilor prin intermediul pointerilor.
Atunci când o functie normala (care nu este virtuala) este definita într-o clasa de baza si redefinita în clasele derivate, la apelul acesteia ca functie membra a unui obiect pentru care se cunoaste un pointer, se selecteaza functia dupa tipul pointerului, indiferent de tipul obiectului al carui pointer se foloseste (obiect din clasa de baza sau obiect din clasa derivata).
Daca o functie este definita ca functie virtuala în clasa de baza si redefinita în clasele derivate, la apelul acesteia ca functie membra a unui obiect pentru care se cunoaste un pointer, se selecteaza functia dupa tipul obiectului, nu al pointerului. Sunt posibile mai multe situatii:

- Daca obiectul este din clasa de baza nu se poate folosiun pointer la o clasa derivata.
- Daca obiectul este de tip clasa derivata si pointerul este pointer la clasa derivata, se selecteaza functia redefinita în clasa derivata respectiva.
- Daca obiectul este de tip clasa derivata, iar pointerul folosit este un pointer la o clasa de baza a acesteia, se selecteaza functia redefinita în clasa derivata corespunzatoare tipului obiectului.

Acesta este mecanismul de virtualitate si el permite implementarea polimorfismului în clasele derivate. O functie redefinita într-o clasa derivata *domina* functia virtuala corespunzatoare din clasa de baza si o înlocuieste chiar daca tipul pointerului cu care este accesata este pointer la clasa de baza. În apelul ca functie membra a unui obiect dat cu numele lui, functiile virtuale se comporta normal, ca functii redefinite.
Nu pot fi declarate functii virtuale functiile nemembre, constructorii, functiile friend. Functia declarata `virtual` în clasa de baza actioneaza ca o descriere generica prin care se defineste interfata comuna, iar functiile redefinite în clasele derivate precizeaza actiunile specifice fiecarei clase derivate asa cum se va ilustra în exercitiul urmator.

Exercitiul 2:

Entitatile cerc, dreptunghi, triunghi sunt corelate între ele prin aceea ca toate sunt forme geometrice, deci ele au în comun conceptul de forma geometrica. Pentru a reprezenta un *cerc*, *triunghi* sau *dreptunghi* într-un program, trebuie ca aceste clase, care reprezinta fiecare o forma geometrica în parte împreuna cu proprietatile sale, sa aiba în comun clasa care reprezinta în general o *forma geometrica*. Sa se defineasca o clasa de baza `Shape` care are ca membri : centrul figurii, aria figurii si o functie de desenare `display`.

```
class Shape{
protected:
      int centerx, centery;
public:
      Shape();
      ~Shape();
      double aria(){}
      void display();
};
```

Derivati din aceasta clasa, corespunzator pentru cele trei figuri geometrice, clasele `Cerc`, `TriunghiEchilateral` si `Patrat`, si completati pentru fiecare clasa în parte datele membre necesare,

constructorii, destructorii, functia de calcul al ariei si functia `display` care va afisa numele figurii de desenat. În functia `main()` creati câte un obiect cerc, triunghi echilateral si patrat si afisati aria fiecaruia; apoi declarati un vector de 3 pointeri catre clasa `Shape`. Folositi acesti pointeri pentru a crea dinamic câte un obiect de tipul claselor derivate si afisati pentru fiecare obiect în parte aria si tipul sau.

2. Mostenirea functiilor virtuale

Atributul `virtual` se mosteneste pe tot parcursul derivarii. Cu alte cuvinte, daca o functie este declarata de tip `virtual` în clasa de baza, functia redefinita într-o clasa derivata este functie de tip `virtual` în mod implicit (fara sa fie nevoie de declaratie explicita `virtual`). Acest lucru înseamna ca, daca aceasta clasa derivata serveste pentru definirea unei noi derivari, functia va fi mostenita de asemenea de tip `virtual`. Pot sa apara si alte situatii în mostenirea functiilor virtuale. De exemplu, daca o functie virtuala nu este redefinita într-o clasa derivata, atunci se foloseste functia mostenita din clasa de baza si pentru obiecte de tip clasa derivata.

Exercitiul 3:

Reluati exercitiul anterior si derivati din clasa `Cerc` o noua clasa, `CercColorat`, care sa contina, în plus, culoarea cercului data printr-un numar întreg iar din clasa `Patrat` derivati o noua clasa `Cub`. În functia `main()` creati în mod dinamic un obiect tip `CercColorat` si un obiect `Cub` si afisati ariile celor doua obiecte.

3. Destructori virtuali

Destructorii pot fi declarati de tip `virtual`, si aceasta declaratie este absolut necesara în situatia în care se dezaloca un obiect de tip clasa derivata folosind pointer la o clasa de baza a acesteia.

Exercitiul 4:

Completati programele de la exercitiile 2 si 3 cu instructiuni de afisare a tipului destructorilor si cu instructiuni de stergere a obiectelor create. Ce mesaje sunt afisate la rularea programelor?

Mesajele afisate evidenteaza faptul ca la stergerea unui obiect tip clasa derivata creat prin pointer la clasa de baza a fost sters numai un subobiect al acestuia, subobiectul din clasa de baza. Acest lucru face ca memoria libera (heap) sa ramâna ocupata în mod inutil (cu partea de date a clasei derivate), desi se intentiona eliminarea completa din memorie a obiectului creat.

Exercitiul 5:

În programele anterioare declarati destructorul din clasa de baza de tip virtual. Ce mesaje se obtin la rularea programelor?

Declararea destructorului din clasa de baza de tip `virtual` rezolva problema ocuparii inutile a memoriei: la stergerea obiectelor se apeleaza destructorul clasei derivate dupa tipul obiectului nu al pointerului folosit. Destructorul clasei derivate elibereaza partea din clasa derivata a obiectului, dupa care (în mod implicit, fara sa fie nevoie ca programatorul sa specifice acest lucru), este apelat destructorul clasei de baza care elibereaza subobiectul de baza din obiectul dat.

4. Clase abstracte

De cele mai multe ori functia declarata de tip `virtual` în clasa de baza nu defineste o actiune semnificativa si este neaparat necesar ca ea sa fie redefinita în fiecare din clasele derivate (de exemplu, functia `aria()` în clasa `Shape`). Pentru ca programatorul sa fie obligat sa redefineasca o functie virtuala în toate clasele derivate în care este folosita aceasta functie, se declara functia respectiva *virtuala pura*. O functie virtuala pura este o functie care nu are definitie în clasa de baza, iar declaratia ei arata în felul urmator:

```
virtual tip_returnat nume_functie(lista_arg) = 0;
```

O clasa care contine cel putin o functie virtuala pura se numeste *clasa abstracta*. Deoarece o clasa abstracta contine una sau mai multe functii pentru care nu exista definitii (functii virtuale pure), nu pot fi create instante (obiecte) din acea clasa, dar pot fi creati pointeri si referinte la astfel de clase abstracte. O clasa abstracta este folosita în general ca o clasa fundamentala, din care se construiesc alte clase prin derivare.
Orice clasa derivata dintr-o clasa abstracta este, la rândul ei clasa abstracta (si deci nu se pot crea instante ale acesteia) daca nu redefinesc toate functiile virtuale pure mostenite. Daca o clasa redefineste toate functiile virtuale pure ale claselor ei de baza, devine clasa normala (ne-abstracta) si pot fi create instante (obiecte) ale acesteia.

Exercitiul 6:

Modificati programele anterioare astfel încât clasa `Shape` sa fie abstracta. Ce modificari trebuie aduse programelor pentru obtinerea unei functionari corecte?

5. Un caz practic

Un exemplu practic care evidentiaza utilitatea functiilor virtuale în general si a functiilor virtuale pure în special este prezentat în continuare:

```
class Convert{
protected:
      double x;         // valoare intrare
      double y;         // valoare iesire
public:
      Convert(double i){x = i;}
      double getx(){return x;}
      double gety(){return y;}
      virtual void conv() = 0;
};
// clasa FC de conversie grade Farenheit in grade Celsius
class FC: public Convert{
public:
      FC(double i):Convert(i){}
      void conv(){y = (x-32)/1.8;}
};
// clasa IC de conversie inch in centimetri
class IC: public Convert{
public:
      IC(double i):Convert(i){}
      void conv(){y = 2.54*x;}
};
void main (){
      Convert* p = 0;         // pointer la baza
```

```
        cout<<"Introduceti valoarea si tipul conversiei: ";
        double v;
        char ch;
        cin >> v >> ch;
        switch (ch){
        case 'i':               //conversie inch -> cm (clasa IC)
              p = new IC(v);
              break;
        case 'f':      //conv. Farenheit -> Celsius (clasa FC)
              p = new FC(v);
              break;
        }
        if (p){
              p->conv();
              cout << p->getx() << "---> " << p->gety()<< endl;
              delete p;
        }
}
```

Acest program converteste date dintr-o valoare de intrare într-o valoare de iesire (din grade Farenheit în grade Celsius, din inch în centimetri). Clasa de baza absrtracta `Convert` este folosita pentru crearea prin derivare a câte unei clase specifice fiecarui tip de conversie de date dorit. Aceasta clasa defineste datele comune, necesare oricarui tip de conversie, de la o valoare de intrare `x` la o valoare de iesire `y`. Functia de conversie `conv()` nu se poate defini în clasa de baza, ea fiind specifica fiecarui tip de conversie în parte; de aceea se declara functie virtuala pura si trebuie sa fie redefinita în fiecare clasa derivata. În functia `main()` se executa o conversie a unei valori introduse de la consola, folosind un tip de conversie (o clasa derivata) care se selecteaza pe baza unui caracter introdus la consola. Acesta este un exemplu care pune în evidenta necesitatea functiilor virtuale: deoarece nu se cunoaste în momentul compilarii tipul de conversie care se va efectua, se foloseste un pointer la clasa de baza pentru orice operatie (crearea unui obiect de conversie nou, apelul functiei `conv()`, afisarea rezultatelor, distrugerea obiectului la terminarea programului). Singura diferentiere care permite selectia corecta a functiilor, este tipul obiectului creat, care depinde de tipul conversiei cerute la consola.

6. Polimorfismul

Una din caracteristicile importante ale programarii orientate pe obiecte este aceea ca permite definirea unei interfete comune pentru mai multe metode specifice diferitelor functionalitati. Aceasta comportare, care asigura simplificarea si organizarea sistemelor de programe complexe, este cunosuta sub numele de *polimorfism*. Polimorfismul introdus prin mecanismul de virtualitate este polimorfism la nivel de executie, care permite legarea târzie (*late binding*) între evenimentele din program, în contrast cu legarea timpurie (*early binding*), proprie apelurilor functiilor normale (nevirtuale). Intercorelarea (legarea) timpurie se refera la evenimentele care se desfasoara în timpul compilarii si anume apeluri de functii pentru care sunt cunoscute adresele de apel: functii normale, functii supraîncarcate, operatori supraîncarcati, functii membre nevirtuale, functii `friend`. Apelurile rezolvate în timpul compilarii beneficiaza de o eficienta ridicata. Termenul de legare târzie se refera la evenimente din timpul executiei. În astfel de apeluri, adresa functiei care urmeaza sa fie apelata nu este cunoscuta decât în momentul executiei programului.

Functiile virtuale sunt evenimente cu legare târzie: accesul la astfel de functii se face prin pointer la clasa de baza, iar apelarea efectiva a functiei este determinata în timpul executiei de tipul obiectului indicat, pentru care este cunoscut pointerul la clasa sa de baza. Avantajul principal al legarii târzii (permisa de polimorfismul asigurat de functiile virtuale) îl constitue simplitatea si flexibilitatea programelor rezultate. Dezavantajul cel mai important este timpul suplimentar necesar selectiei functiei apelate efectiv din timpul executiei, ceea ce conduce la un timp de executie mai mare al programelor.