11. Funcţii virtuale. Clase de bază virtuale

11.1 Funcţii virtuale

O funcţie virtuală este o funcţie membră specială, invocată printr-o referinţă sau un pointer la o anumită clasă de bază (publică), funcţie care este definită şi în clasele derivate. Dintre toate funcţiile virtuale cu acelaşi nume, cea care este efectiv apelată se determină doar în momentul execuţiei pe baza tipului actual al obiectului invocator. Fenomenul este cunoscut sub numele de *legare dinamica*.

Exemplu:

```
void ecranFinal( AnimalZoo *pz )
{
   // lista selectata de animale
   for( AnimalZoo *p=pz; p; p=p->next )
      p->draw();
}
```

Legarea dinamica încapsulează detaliile ierarhiei de derivare. Ea poate fi modificată fără a modifica funcţiile deja scrise. O funcţie virtuală se declară prin cuvântul cheie `virtual` plasat în faţa declaraţiei sale (nu şi a definiţiei).

Exemplu:

```
class C
{
   public:
      virtual int bar(); // declaratie
...
};
int C::bar() { ... } // definitie
```

Există funcţii virtuale a căror definiţie nu este necesară într-o clasă de bază:

```
class C
{
   public:
      virtual void draw();
      virtual void info();
...
};
```

Proiectantul unei clase poate indica faptul că o funcţie virtuală este nedefinită pentru o anumită clasă, iniţializând declaraţia ei cu 0.

Exemplu:

```
class AnimalZoo
{
   public:
      virtual void draw() = 0;
...
};
```

Funcţiile virtuale care nu sunt definite pentru tipul de bază se numesc funcţii virtuale pure. O clasă care conţine funcţii virtuale pure poate fi folosită doar ca şi clasă de bază pentru derivari ulterioare

Exemplu:

```
AnimalZoo az;   // eroare
```

O clasă care declară pentru prima dată în ierarhia de derivare o funcţie virtuală trebuie fie să o definească, fie să o declare pură. Dacă o defineşte, atunci clasele derivate pot să nu o redefinească. Dacă o declară pură, atunci clasele derivate trebuie fie să o definească, fie să o declare din nou pură.

- Clasele derivate pot redefini funcţiile virtuale moştenite şi îşi pot defini propriile funcţii virtuale.

Exemplu:

```
class Urs: public AnimalZoo
{
   public:
      void draw(); // redefineste mostenirea de la AnimalZoo
      virtual hiberneaza() { return 1;}
   private:
      void info();
...
};
Urs u;
AnimalZoo& az=u;
az.draw(); // se invoca Urs::draw();
```

- La redefinirea unor funcţii virtuale, cuvântul cheie `virtual` nu mai este obligatoriu (funcţia fiind virtuală pentru întreaga ierarhie), iar numele, semnătura şi tipul trebuie să coincidă cu cele din prima definiţie.

- Nivelul de acces al unei funcţii virtuale este determinat de tipul pointerului sau referinţei la clasa prin care se face explicit invocarea ei.

Exemplu:

```
Urs u;
AnimalZoo& az=u;
az.draw();      // Urs::draw()
az.info();      // Urs::info()
```

Atentie !

Deşi `Urs::info` este o funcţie privată, ea poate fi invocată direct prin intermediul unei referinţe la `AnimalZoo`, deoarece în această clasă, `info()` era publică.

Să presupunem că lista de `AnimalZoo` transmisă funcţiei `ecranFinal` nu mai e necesară după execuţia ei. Deci în mod firesc, avem nevoie de o secvenţă de forma:

Exemplu:

```
for( AnimalZoo *p=pz->next; p; pz=p, p=p->next )
   delete pz; // determina apel de destructor ~AnumalZoo; insuficient.
```

- Pentru apelarea destructorului tipului invocator într-o ierarhie de derivare, destructorii vor fi declaraţi virtuali. (Destructorii se moştenesc !)

Exemplu:

```
class AnimalZoo
{
   public:
      virtual ~AnimalZoo();
...
};
```

11.2 Clase de bază virtuale

Deăi o clasă de bază poate apărea doar o singură dată într-o listă de derivare, totuşi este posibil ca în ierarhia de derivare să apară de mai multe ori.

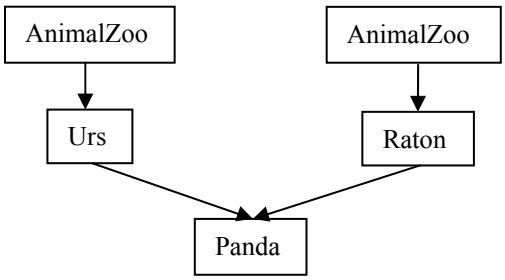
Exemplu:

```
class AnimalZoo
{
   public:
      AnimalZoo() { zona=0; name=0; }
      AnimalZoo( char*, short );
      locate();
```

```
   protected:
      short zona;
      char *name;
};
class Urs : public AnimalZoo { ... };
class Raton : public AnimalZoo { ... };
class Panda : public Urs, public Raton { ... };
```

Prin mecanismul implicit de moştenire, o clasă derivată conține toți membrii claselor de bază, fapt care generează imposibilitatea selectării membrilor claselor care apar de mai multe ori în ierarhie.

Exemplu:



Moştenire multiplă nevirtuală

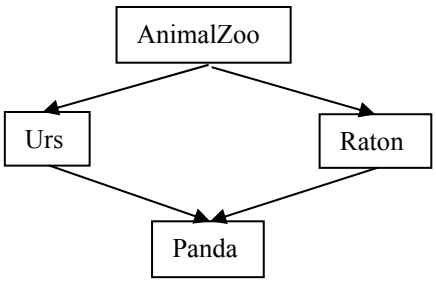
```
Panda p;
p.name; // ambiguitate, nu se poate accesa direct pentru ca nu se
        // poate face distinctie intre cele 2 instante al lui name.
```

Mecanismul implicit de moştenire poate fi ocolit prin intermediul unor clase de bază partajate: ori de câte ori o astfel de clasă ar apărea într-o ierarhie, ea va fi moştenită o singură dată. În C++, clasele de bază partajate se indică prin cuvântul cheie `virtual` pus înaintea numelui clasei în lista de derivare.

Exemplu:

```
class Urs : public virtual AnimalZoo { ... };
class Raton : public virtual AnimalZoo { ... };
```

Reprezentarea grafică a ierarhiei de moştenire ar arăta astfel:



Observație:

- Unele implementări de C++ solicită prezența unui constructor implicit în clasele virtuale (sau măcar a unui constructor cu valori implicite pentru toate argumentele). În rest, clasele virtuale au aceeaşi definiție ca şi clasele de bază obişnuite.

În mod obişnuit, o clasă derivată poate iniția explicit numai clasa de bază din lista de derivare (nivel ierarhic imediat superior). Clasele virtuale fac excepție de la această regulă. O clasă virtuală se inițializează de "cea mai derivată clasă din ea". În cazul nostru, `Panda` poate iniția explicit `AnimalZoo` în lista de inițializare a constructorului. Dacă nu se inițializează explicit `AnimalZoo`, va fi invocat constructorul implicit `AnimalZoo()`.

Initializarile `AnimalZoo` ale lui `Raton` sau `Urs` nu sunt aplicate niciodată porțiunii `AnimalZoo` dintr-un obiect `Panda`.

Exemplu:

```
Raton :: Raton( char *urm ) : AnimalZoo( nm, RATON ) { ... }
Urs   :: Urs ( char *nm )   : AnimalZoo( nm, URS )   { ... }
```

Constructor pentru Panda

```
Panda :: Panda( char *nm )
: AnimalZoo( nm, PANDA ), Urs( nm ), Raton( nm )
{ ... }
// AnimalZoo din Urs si Raton nu se mai invoca
```

Dacă definim constructorul astfel

```
Panda :: Panda( char *nm )
   : Urs( nm ), Raton( nm )
{ ... }
// se invoca constructorul implicit din AnimalZoo
```

Constructorii claselor virtuale sunt invocați totdeauna înaintea constructorilor claselor de bază obişnuite.

Exemplu:

```
class UrsJucarie : public Urs, public virtual AnimalJucarie
{
...
};
UrsJucarie Teddy;
```

Constructorii invocați sunt:

```
AnimalJucarie();        // clasa de baza virtuala
AnimalZoo();            // clasa de baza pentru Urs
Urs();                  // clasa de baza nevirtuala
UrsJucarie();           // constructorul clasei curente
```

Constructorii claselor virtuale se invocă în ordinea identificării acestora în ierarhia de moştenire.

Exemplu:

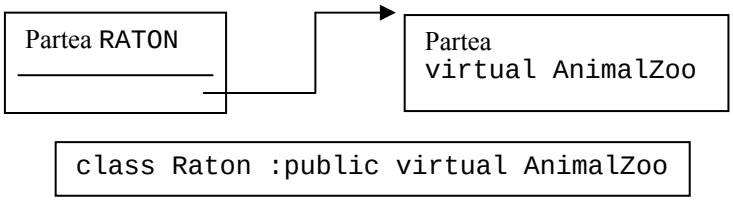
```
class Panda : public Ocrotit, public Ierbivor,
              public Raton, public Urs
{
...
};
Panda Beep;
```

Această definiție determină invocarea următorilor constructori:

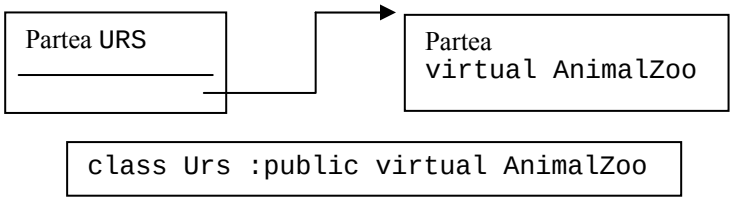
```
// constructorii claselor virtuale de baza pentru Ocrotit, Ierbivor,
// Raton, Urs
AnimalZoo();
// constructorii claselor de baza, in ordinea declararii lor
Ocrotit();
Ierbivor();
Raton();
Urs();
// constructorii clasei curente
Panda();
```

11.3 Accesarea membrilor claselor virtuale

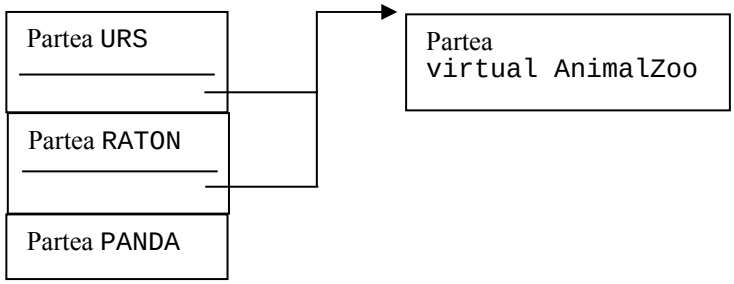
Când într-o ierarhie de derivare o clasă apare atât ca virtuală cât şi ca nevirtuală, se va moşteni o dată pentru toate aparițiile virtuale şi de fiecare dată pentru fiecare apariție nevirtuală.




```
class Raton :public virtual AnimalZoo
```




```
class Urs :public virtual AnimalZoo
```




```
class Panda :public Urs, public: Raton
```

Intern, membrii moşteniţi virtual sunt adresaţi prin pointeri , transparent utilizatorului. Membrii moşteniţi din clasa virtuală suportă aceleaşi reguli de acces ca şi în urma unei derivări nevirtuale. Ce se întâmplă însă când `Panda` include atât o instanţă publică cât şi o instanţă privată a unei clase de bază virtuale ?

Exemplu:

```
class AnimalZoo
{
   public:
      void locate();
   protected:
      short zona;
...
};
class Urs   : public virtual AnimalZoo  { ... };
class Raton : private virtual AnimalZoo { ... };
class Panda : public Urs, public Raton  { ... };
```

Pentru că `AnimalZoo` este o clasă de bază virtuală, `Panda` moşteneşte o singură copie a membrilor `zona` şi `locate();` Întrebarea care se pune este dacă `Panda` are sau nu acces la aceşti membri. Răspunsul este şi DA şi NU, şi anume: DA - de-a lungul caii de derivare publice şi NU - de-a lungul căii de derivare private.

Exemplu:

```
Panda p;
p.locate(); // corect, predomina calea derivarii publice
```

În general, oricâte clase de bază virtuale există în ierarhie, dacă există cel puţin o instanţă publică, atunci instanţa partajată este considerată publică. În derivarea virtuală funcţiile membre din cele mai derivate clase le domina pe celelalte.

Exemplu:

```
// Presupunem ca in clasa Urs apare membrul void locate().
Urs u;
```

```
u.locate();     // Urs::locate()
Panda p;
p.locate(); // ambiguu in cazul derivarii nevirtuale: Urs::locate()
            // sau Raton::locate() ?
            // Urs::locate() in cazul derivarii virtuale.
```