VI. Metode virtuale

Ce sunt metodele virtuale

- orice metodă definită într-o clasă de bază poate fi redefinită într-o clasă derivată
- un pointer spre clasa de bază ne poate indica un obiect dintr-o clasă derivată
- am vrea să putem apela metoda potrivită
  - în funcţie de clasa de care aparţine cu adevărat obiectul spre care indică pointerul
- acesta este rolul metodelor virtuale

Exemplu 1

```
class A {
public:
    void afisare() {printf("A\n");}
};
class B: public A {
public:
    void afisare() {printf("B\n");}
};
```

Exemplu 1 (cont.)

```
A a;
B b;
A *pa=&b;
pa->afisare();
```

- rezultat afişat: `A`
  - deoarece pa este pointer către clasa A
  - deci se apelează metoda afisare a clasei A

Exemplu 2

```
class A {
public:
    virtual void afisare() {printf("A\n");}
};
class B: public A {
public:
    void afisare() {printf("B\n");}
};
```

Exemplu 2 (cont.)

```
A a,*p;
B b;
a.afisare();
b.afisare();
p=&a;
p->afisare();
p=&b;
p->afisare();
```

Exemplu 2 (cont.)

- rezultate afişate

```
A
B
A
B
```

Interpretare

- când apelul se face prin intermediul obiectelor a şi b - acelaşi comportament ca la metodele obişnuite
- când apelul se face prin intermediul pointerului pa - se apelează metoda corespunzătoare clasei de care aparţine obiectul
  - este A sau B, după caz
  - deşi p este pointer spre clasa A

Cum se face? (1)

- pointerul p primeşte valori la momentul execuţiei
- abia atunci se vede cărei clase aparţine obiectul indicat de pointer
- deci nu compilatorul decide metoda apelată
- trebuie să avem disponibile informaţii suplimentare la momentul execuţiei
  - ca să putem decide ce metodă trebuie apelată

Cum se face? (2)

```
class A {
    int a;
public:
    virtual void afisare()
    {...}
};
```

```
class B: public A {
    int b;
public:
    void afisare()
    {...}
};
```

Cum se face? (3)

```
printf("A\t%d\n",sizeof(A));
printf("B\t%d\n",sizeof(B));
```

- rezultate afişate

```
A    8
B    12
```

- apar 4 octeţi suplimentari

VMT (1)

- este un membru suplimentar
- pointer către un tablou
  - VMT = *Virtual Method Table*
  - elementele tabloului - pointeri
  - câte unul către fiecare metodă virtuală a clasei
- apelul unei metode virtuale
  - se preia din VMT pointerul spre metoda dorită şi se face apelul metodei respective
  - la execuţie, nu la compilare

VMT (2)

- pointerul către VMT este plasat la începutul obiectului
  - deplasament 0
  - abia apoi urmează membrii de date
- clasa derivată nu moşteneşte şi acest membru din clasa/clasele de bază
  - are propriul VMT
- VMT este utilizat la apelul metodelor virtuale numai prin pointeri sau referinţe

Apel din limbaj de asamblare

```
B x;
A *pa=&x;
_asm {
   mov eax,pa              //(1)
   mov ebx,[eax]           //(2)
   mov ecx,pa              //(3)
   call dword ptr [ebx]    //(4)
}
```

Detaliere

- linia 1
  - preluare pointer pa în registrul eax
- linia 2
  - preluare pointer către VMT în registrul ebx
- linia 3
  - plasare pointer *this* în registrul ecx
- linia 4
  - apelul metodei afisare; adresa metodei - primul element din VMT

Moştenire virtuală (1)

```
class A {
   int a;
};

class B1: public A {
   int b1;
};
```

```
class B2: public A {
   int b2;
};

class C: public B1,
   public B2 {
   int c;
};
```

Moştenire virtuală (2)

- un obiect din clasa C include două obiecte din clasa A
  - unul moştenit prin clasa B1
  - celălalt moştenit prin clasa B2
- uneori dorim să avem un singur obiect din clasa A
- apelăm la moştenirea virtuală

Moştenire virtuală (3)

```
class A {...};
class B1: virtual public A {...};
class B2: virtual public A {...};
class C: public B1, public B2 {...};
```

- trebuie ca atât B1, cât şi B2 să moştenească virtual clasa A

Structura clasei (1)

- unde se găseşte acum obiectul din clasa A în cadrul obiectului din clasa C?

```
printf("%d\n",sizeof(C));
```

- dacă nu foloseam moştenirea virtuală, rezultatul ar fi fost 20
- rezultat afişat: `24`
- de ce?

Structura clasei (2)

- deplasament 0: obiectul moştenit de tip B1
- deplasament 8: obiectul moştenit de tip B2
- deplasament 16: membrul c
- deplasament 20: obiectul moştenit de tip A

- de ce obiectele de tip B1 şi B2 au câte 8 octeţi fiecare?

Structura clasei (3)

- adăugăm clasei A încă un membru de tip int
- cu cât creşte dimensiunea unui obiect din clasa C?
- ne aşteptăm să crească cu 12 octeţi
  - 4 octeţi - obiectul moştenit de tip B1
  - 4 octeţi - obiectul moştenit de tip B2
  - 4 octeţi - obiectul moştenit de tip A
- dar dimensiunea creşte doar cu 4 octeţi

Structura clasei (4)

- de fapt, şi clasele B1 şi B2 şi-au modificat structura prin moştenirea virtuală
- primii 4 octeţi nu reprezintă spaţiu pentru obiectul moştenit din clasa A
- este un pointer către o structură de date
  - cu ajutorul său putem determina deplasamentul obiectului moştenit din clasa A
  - la momentul execuţiei