Object Oriented Programming in Director

Conf.dr. Marian DÂRDALĂ, prof.dr. Ion SMEUREANU, lect.dr. Adriana REVEIU
Catedra de Informatică Economică, ASE Bucureşti

Director is one of the most popular authoring software. As software for developing multimedia applications, Director is an object oriented programming environment. A very important issue to develop multimedia applications is the designing of their own classes. This paper presents the particular aspects concerning the available facilities offered by Lingo to design classes and to generate objects.
Keywords: *class, object, encapsulation, inheritance, script*

Introducere

Conceptele şi proprietăţile ce ţin de programarea orientată obiect sunt implementate în mod diferit în mediile de programare. De exemplu, în *C++* se permite moştenirea multiplă în timp ce în *C#* sau *Java* acest lucru nu este permis. Pe de altă parte, în *C#* sau *Java* există implementat conceptul de interfaţă în timp ce acesta lipseşte din implementarea *C++*. Diferenţele de implementare a conceptelor legate de programarea orientată obiect sunt mai vizibile cînd acestea sunt implementate în limbajele de tip script. Astfel, în *Lingo* (limbajul scriptic al produsului *Director*) conceptele legate de programarea orientată obiect sunt simplificate în sensul că, nu se permite supraîncărcarea operatorilor, definirea de clase abstracte, de interfeţe etc. Proprietăţile programării orientate obiect implementate în Lingo se referă, în mod deosebit, la încapsulare şi moştenire.
Indiferent de implementare, cu ajutorul obiectelor se pot crea noi entităţi capabile să eficientizeze dezvoltarea aplicaţiilor în sensul dorit de programator.

Încapsularea

Este bine cunoscut că încapsularea este o proprietate a programării orientate obiect care presupune ca datele şi codul aferent datelor respective să coexiste într-o structură unică denumită clasă. In *Director* nu există o entitate sintactică a limbajului *Lingo* care să permită definirea unei clase, ea se defineşte utilizîndu-se un script de tip *parent script* care se va stoca în mod corespunzător în componenta *Cast*.
O clasă conţine date, care în mod uzual se definesc în legătură cu tipul lor (de exemplu: *int a* sau *float x*). In *Lingo* variabilelor nu li se asociază tipuri de date, acestea derivă din contextul de utilizare (de exemplu, în expresia *a + b*, *a* şi *b* sunt de tip numeric pentru că operatorul + se aplică valorilor numerice în timp ce în expresia *x & y*, *x* şi *y* sunt variabile de tip şir de caractere pentru că operatorul & se aplică şirurilor de caractere în sensul concatenării lor). Deoarece în *Lingo* nu se asociază tipuri variabilelor, într-o clasă datele membre se precizează prin intermediul cuvîntului cheie *property* (de exemplu *property a, b*).
Referitor la metode, acestea pot primi parametri şi pot întoarce rezultate. O metodă specială care are rolul de a iniţializa datele membre ale unui obiect şi de a genera obiecte se numeşte *constructor*; în *Lingo* el are numele predefinit *new*.

```
property a
on new me, xx
      a=xx
      return me
end
```

Proprietatea *a* a fost iniţializată din parametrul de apel al constructorului (*xx*) iar *return me* are ca scop generarea obiectului identificat prin *me*. Parametrul *me* (cuvînt cheie al limbajului) este similar cu *this* din limbajul *C#* sau *C++* şi referă obiectul curent.
Dacă se doreşte definirea unei metode pentru a furniza valoarea proprietăţii *a*, atunci ea poate avea forma:

```
on get_a
      return a
end
```

iar pentru a modifica valoarea parametrului *a* din parametrul de apel, metoda se poate defini în forma:

```
on set_a me, t
      a=t
end
```

Este important de reținut faptul că, atunci cînd o metodă primeşte parametri, primul parametru trebuie să fie *me,* după care se indică şi ceilalți parametri. Membrii clasei se pot invoca şi prin obiectul curent (*me*), astfel:

```
on set_a me, t
      me.a=t
end
```

Definirea obiectelor în Director

Un obiect în *Director* se crează prin apelul constructorului care are ca scop alocarea spațiului de memorie şi reținerea adresei respective într-o variabilă prin intermediul căreia se vor accesa proprietățile şi metodele obiectului respectiv. In *Lingo* clasa se identifică prin numele parent script-ului care memorează definiția clasei. Dacă numele scriptului este *clasa_mea*, atunci generarea obiectului se face în forma:

```
ob = script("clasa_mea").new(100)
```

proprietatea *a* ia valoarea 100.

Accesarea unui membru al obiectului se face printr-o construcție sintactică asemănătoare cu cea cunoscută din limbajele de programare orientate obiect. In exemplul prezentat dacă se dorește accesarea proprietății *a*, atunci expresia de referire este *ob.a*:

```
vint = ob.a
put vint
```

Cînd membrul este o metodă, referirea se face în acelaşi mod, doar că se sugerează prin *( )* apelul ei:

```
ob.set_a(200)
vint=ob.get_a()
put vint
```

Încapsularea tipului media

In *Director* se definesc obiecte pentru a crea noi entități, cu semnificații particulare, ce țin de scopul unei aplicații. Astfel, dacă se consideră o aplicație care are ca scop colorarea județelor țării în funcție de valoarea ratei şomajului, atunci un județ se reprezintă în *Director* ca un obiect de tip imagine (*bitmap*). Existînd sub forma unui obiect de tip imagine el are proprietăți şi comportamente specifice, deja definite. Din punctul de vedere al aplicației județul trebuie să aibă proprietăți specifice cum ar fi numărul de şomeri, populația activă precum şi metode specifice cum ar fi cele pentru colorarea județului în funcție de rata şomajului, resetarea culorii etc.

In această abordare, județul trebuie să existe ca un obiect unic care să aibă proprietăți şi metode ce-l exploatează din punct de vedere grafic dar şi proprietăți şi metode ce țin de semnificația aplicației.

Incapsularea elementelor într-o singură clasă se face din punct de vedere al obiectului grafic prin definirea unei proprietăți care va referi canalul (*sprite*-ul) pe care va fi instanțiat județul respectiv. Datele cu privire la culoarea utilizată (*cul*) pentru colorarea județului precum şi intervalele de valori definite prin (limita minimă – *li* şi maximă – *ls* a ratei şomajului) se vor stoca într-o listă de proprietăți (vector de structuri – *intervale*), în forma:

```
it1=[#li:0, #ls:10, #cul:96]
it2=[#li:10, #ls:20, #cul:97]
it3=[#li:20, #ls:100, #cul:98]
intervale=[it1,it2,it3]
```

Clasa *județ* se definește în forma:

```
-- populația activă
property pa
-- numărul de şomeri
property ns
-- numărul canalului pe care se află instanțiat județul
property nsp
-- vectorul de intervale
property interv
-- definirea constructorului
on new me, fpa, fns, finterv, fnsp
  pa=fpa
  ns=fns
  nsp=fnsp
  interv=finterv
  return me
end me
-- metoda de colorare a judetului
on coloreaza_judet me
-- rata somajului calculată procentual
  rs=ns*100/pa
  repeat with i=1 to interv.count
    if rs>=interv[i].li and
rs<interv[i].ls then
```

-- colorarea judetului cu culoarea corespunzatoare

```
sprite(nsp).backcolor=interv[i].cul
      exit repeat
    end if
  end repeat
end
```

-- metoda de resetare a culorii județului

```
on reseteaza me
```

-- colorarea judetului cu culoarea alba

```
  sprite(nsp).backcolor=0
end
```

Considerînd că *parent script*-ul care stochează clasa se numeşte *judet*, se vor genera obiectele sub forma unui vector de obiecte (*jud*). Datele referitoare la județe, cu care se vor inițializa obiectele, se vor prelua din fişierul text *date.txt*.

```
 jud=[]
 pf=new xtra("fileio")
 pf.openfile("date.txt",1)
 lc=pf.readline()
 i=1
 repeat while lc<>EMPTY
    repeat with j=1 to nj
      if
sprite(j).member.name=lc.word[1]
then
        nsp=j
        exit repeat
      end if
    end repeat

jud[i]=script("judet").new(lc.word[2
], lc.word[3], intervale, nsp)
    lc=pf.readline()
    i=i+1
 end repeat
 pf.closefile()
```

Pentru procesarea hărții se vor apela metodele:

- de colorare:

```
repeat with i=1 to jud.count
    jud[i].coloreaza_judet()
end repeat
```

- de resetare a culorilor:

```
repeat with i=1 to jud.count
    jud[i].reseteaza()
end repeat
```

In figura 1 se ilustrează modul de afişare a rezultatului şi interfața grafică.

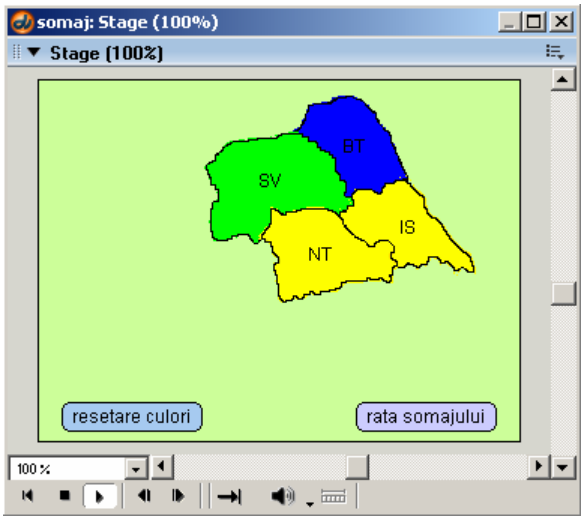

Fig.1 Interfața grafică a aplicației

Moştenirea

Proprietatea de moştenire în *Director*, ca şi în alte medii de dezvoltare a aplicațiilor, în manieră orientată obiect este utilă pentru a construi clase ca extensii ale altora deja existente. In *Lingo* moştenirea se stabileşte prin declararea unei proprietăți denumită *ancestor* prin intermediul căreia se referă proprietățile şi metodele clasei de bază. Considerînd clasa de bază ca fiind parent scriptul definit la punctul 2 avînd numele *baza*, clasa derivată se defineşte în *parent script*-ul *derivat* în forma:

```
property der
property ancestor
on new me, yy, zz
-- creare obiect de bază
  ancestor=script("baza").new(yy)
  der=zz
  return me
end
on get_add_prop
  return der+ancestor.a
end
```

unde, metoda `get_add_prop` returnează rezultatul adunării dintre *der*, proprietatea clasei derivate şi *a* proprietatea clasei de bază. Se observă că, proprietatea din clasa de bază a fost referită prin proprietatea *ancestor*. O altă modalitate de referire a unei proprietăți din clasa de bază din clasa derivată constă în utilizarea lui *me* în forma:

```
on get_add_prop me
  return der+me.a
end
```

Generarea obiectelor de clasă derivată precum şi referirea proprietăţilor sau a metodelor fie din clasa de bază, fie din clasa derivată se exemplifică prin secvenţa:

```
obd=script("derivat").new(1000,
7000)

varder=obd. get_add_prop ()
put varder
varder=obd.get_a()
put varder
```

Concluzii

Utilizarea conceptelor specifice programării orientate obiect, în produsele de creaţie multimedia, oferă un mod elegant şi eficient de dezvoltare a aplicaţiilor. Definirea de clase care să încapsuleze proprietăţi obişnuite cu cele care referă resurse multimedia precum şi ataşarea de noi comportamente constituie argumente în favoarea dezvoltării de aplicaţii modularizate avînd la bază aceste entităţi.

Bibliografie

⇒ Nyquist, R. J., Martin, R. (2000). — *Director 8 and Lingo Bible*, IDG Books Inc.
⇒ * * * — (2004). *Director MX 2004 – Director Scripting Reference,* Macromedia Inc.
⇒ * * * — (2004). *Director MX 2004 – Using Director,* Macromedia Inc.
⇒ * * * — (2001). *Macromedia Director 8.5 Shockwave Studio – What's New in Director Shockwave Studio,* Macromedia Inc.
⇒ http://www.fbe.unsw.edu.au/learning/director/
⇒ http://brennan.young.net/Edu/Lingvad.html#LessonTOC