https://laurentbloch.net/BlogLB/Programmer-des-objets-avec-Bigloo

Programmer des objets avec Bigloo

- Zinformatiques - L'informatique : science et industrie - Scheme et les langages Lisp -

Date de mise en ligne : jeudi 4 août 2005

Afin de s'adonner à la programmation par objets, les langages de la famille *Lisp*, dont *Scheme*, se sont dotés d'extensions adaptées, qui se conforment plus ou moins toutes au modèle CLOS *(Common Lisp Object System)*. Cet article en donne un exemple avec le système d'objets de Bigloo, la version de Scheme que j'utilise pour mes cours.

Connaissez-vous le livre de Matthias Felleisen et Daniel P. Friedman intitulé A Little Java, A Few Patterns ? C'est sans doute une des meilleures introductions possibles au monde de la programmation par objets, au langage Java et aux *Design Patterns* (patrons de conception ? patrons de méthode ?), c'est à la fois facile, profond et amusant. Si jusqu'à ce jour vous avez été allergique à Java, achetez ce livre et vous verrez une approche tout à fait originale de l'enseignement de la programmation, qu'une lectrice n'a pas hésité à qualifier de *socratique* : un programmeur chevronné exhibe des textes de programmes, un novice pose des questions et essaie de mettre les réponses en pratique.
Aucune règle n'est formulée, aucune définition n'est stipulée, elles sont seulement inférées des commentaires sur le programme en construction.

La trame du livre de Felleisen et Friedman tourne autour d'histoires de brochettes et de pizzas : on part d'objets très simples, la pâte d'une pizza ou la pique d'une brochette, auxquels viennent s'ajouter des anchois, du fromage, des oignons, des tomates, des morceaux de viande, etc. La combinaison de ces objets permet de décrire une pizza napolitaine ou une brochette végétarienne. Il faut ensuite écrire les prédicats qui permettent de vérifier qu'une brochette est bien végétarienne, ou qu'une pizza ne comporte pas trop d'anchois. Simple, amusant et instructif, ai-je déjà dit. J'avais envie de transposer ces exercices en Scheme pour mes enseignements.

Le modèle objet de Java est assez différent de celui de Bigloo, qui est nettement inspiré de CLOS, et plus précisément de l'interprétation qu'en a donnée Christian Queinnec sous le nom de Meroon [1]. Manuel Serrano (l'auteur de Bigloo) m'a autorisé à publier ici un article sur les objets de Bigloo (il y a aussi une version PostScript) qui constitue une introduction pédagogique à ce système.

Pour Java, les méthodes (c'est-à-dire les programmes qui exécutent les actions demandées aux objets) sont associées aux classes. CLOS, suivi en cela par Meroon et Bigloo, définit les méthodes au moyen de *fonctions génériques*, ce qui donne un aspect assez différent aux programmes, en fait avec Meroon ou Bigloo la syntaxe d'un programme à objets est assez peu différente de celle d'un programme « classique ».

Une conversation avec Manuel m'a permis de préciser les correspondances entre les expressions de Java pour les objets et celles de Bigloo.

Là où Java déclare une classe abstraite et une méthode abstraite (le rôle de la méthode `RemA` est de retirer de la pizza les anchois qui pourraient s'y trouver) :

```
 abstract class PizzaD {
    abstract PizzaD RemA();
}
```

Bigloo définit une classe abstraite et une fonction générique :

```
 (module pizza
   (main theMain)
   (export (abstract-class PizzaD)
       ...
    (generic RemA::PizzaD ::PizzaD)))
       ...
(define-generic (RemA::PizzaD a-pizza::PizzaD))
```

Là où Java dit `new` :

```
 class Cheese extends PizzaD {
    PizzaD p;
    Cheese( PizzaD _p ) {
 p = _p; }

    PizzaD remA() {
 return new Cheese(p.remA()); }
}
```

Bigloo dit `instantiate` :

```
 (module pizza
   (main theMain)
   (export (abstract-class PizzaD)
           (class Crust::PizzaD)
           (class Cheese::PizzaD
              p::PizzaD)
           (class Anchovy::PizzaD
              p::PizzaD)

           (generic remA::PizzaD ::PizzaD)))

(define-generic (remA::PizzaD a-pizza::PizzaD))

(define-method (remA::PizzaD a-pizza::Crust)
   (instantiate::Crust))

(define-method (remA::PizzaD a-pizza::Anchovy)
   (Anchovy-p a-pizza))

(define-method (remA::PizzaD a-pizza::Cheese)
   (instantiate::Cheese
     (p (remA (Cheese-p a-pizza)))))

(define (theMain x)
   (let ((a-pizza (instantiate::Cheese
                     (p (instantiate::Anchovy
                           (p (instantiate::Cheese
                                 (p (instantiate::Crust))
                                 )))))))
      (print "Tout : " a-pizza)
      (print "On retire les anchois : " (remA a-pizza))
      ))
```

Chaque classe qui étend la classe abstraite `PizzaD` possède un attribut `p`, lui-même de type `PizzaD`, qui désigne la pizza sous-jacente en cours de fabrication à laquelle elle vient s'ajouter. Seule la classe `Crust` (pâte) ne contient pas

de champ `p`, évidemment (à moins que l'on envisage des pizzas à double ou triple épaisseur de pâte).

Bref, achetez le livre, vous tirerez tout le sel (surtout avec les anchois) de cet exercice qui n'est enfantin qu'en apparence. Le distributeur pour la France ne semble plus guère s'y intéresser, mais Amazon.co.uk propose de nombreux exemplaires d'occasion à moins de cinq livres sterling.

---

[1] Même s'il en diffère, notamment par le refus d'y incorporer un MOP *(Meta-Object Protocol)* :-)