Outils informatiques

4. Programmation objet en Perl

DESS TEXTE

Introduction

- En programmation orientée objet, on définit des éléments ayant des caractéristiques propres et réalisant certaines opérations. Ces éléments sont ce qu'on appelle des objets.

- On peut voir un objet comme un petit programme autonome ayant ses variables et ses fonctions. Les variables sont appelées des *attributs*(ou propriétés) et les fonctions des *méthodes*.

- Pour utiliser un objet, on doit faire appel à ses méthodes.

- L'intérêt des objets : cacher des comportements complexes derrière une interface "simple" à utiliser.

- En programmation orientée objet : on se concentre sur les données à traiter.

Plan du chapitre

Chapitre 4 : Programmation objet en Perl

1. Notions de Classe, Objet, Attributs et Méthodes
2. Création d'une classe
3. Constructeur et autres méthodes
4. Destructeur
5. Documentation de l'interface
6. Agrégation
7. Héritage

Notions de Classe, Objet, Attributs et Méthodes

- En perl, une classe correspond à un module *spécial* (cf support n°3), elle contient la définition d'un type d'objets (définitions d'attributs et de méthodes),
- un objet est une **référence** à une structure plus ou moins complexe (en général une table de hachage),
- un attribut est un élément de cette structure,
- une méthode est une **procédure** (ou fonction) manipulant cette structure.
- une méthode peut être appelé par une classe (méthode de classe) ou par un objet (méthode d'objet).
- Remarque : à la différence des modules *classiques*, une classe est un espace de nommage protégé.

Création d'une classe

Pour créer une classe, il faut procéder comme suit :

1. choisir le nom de la classe, ce sera le nom du fichier contenant sa définition (avec pour extension **.pm**).

2. définir les propriétés d'un objet de la classe.

3. définir un (ou plusieurs) moyen(s) de créer des objets.

4. définir des opérations assurées par les objets appartenant à la classe (i.e. ses méthodes). Ces méthodes représentent la seule interface que l'on ait avec l'objet (pour manipuler ses attributs notamment).

Constructeur et autres méthodes (1)

- un constructeur est une méthode spéciale renvoyant un objet (i.e. une référence sur une structure),
- pour se faire, on utilise dans le constructeur la fonction **bless()** associant une référence à une classe d'objets,
- l'objet ainsi créé a la particularité de pouvoir appeler des méthodes à partir de lui-même,
- on notera que le constructeur peut avoir n'importe quel nom (en règle général on utilise *new*).

Constructeur et autres méthodes (2)

Ma première classe :

```
#!/usr/bin/perl
use strict;
package Personne;
sub new {
  my($classe,$un_nom,$un_age) = @_;
  my $self = {};
  $self->{NOM} = $un_nom;
  $self->{AGE} = $un_age;
  $self->{PRENOMS} = [];
  bless($self,$classe);
  return($self);
}
1; # pour que la classe soit correctement chargée
```

Le constructeur `new()` est une méthode **de classe** appelée par

```
$moi = Personne->new("Eric",24);
```

Constructeur et autres méthodes (3)

On définit également des méthodes manipulant la structure référencée par l'objet :

```
sub nommer {
  my($Homme,$nouveau_nom) = @_;
  $Homme->{NOM} = $nouveau_nom;
}

sub dire_age {
  my $Homme = shift(@_);
  return ($Homme->{AGE});
}
```

- Ces méthodes sont des méthodes **d'objet** et s'appellent via `$moi->nommer(Jean);`
- La première permet de définir l'attribut *NOM* d'un objet de type `Personne`, la seconde de lire l'attribut *AGE*.
- Dans les méthodes d'objet, le nom de l'objet est le 1er argument, alors que dans les méthodes de classe, c'est le nom de la classe.

Destructeur (1)

- De même qu'un objet est créé, il est détruit après utilisation.
- La méthode *destructeur* est appelée **implicitement** lorsque l'objet disparaît.
- Le nom de cette méthode est imposé : **DESTROY**.
- Le rôle du destructeur est de libérer la mémoire occupée par l'objet.
- Dans de nombreux cas, il n'est pas nécessaire de définir de destructeur car Perl gère la destruction lui-même.
- L'exception est dans le cas des références circulaires : `$moi->{SUIVANT}=$moi;`

Destructeur (2)

Exemple de destructeur :

```
sub DESTROY {
  my $Homme = shift(@_);
  print "Fin de $Homme \n";
}
```

Ici nous affichons un message lorsqu'un objet de type `Personne` est détruit.

Remarque : le desctructeur est une méthode d'objet.

Documentation de l'interface

- Documenter l'interface revient à insérer des commentaires *spéciaux* dans le code.

- Ces commentaires respectent un format spécifique (de type **P**lain **O**ld **D**ocumentation).

- Ils sont alors traités par des programmes dédiés (tels que *pod2man*, *pod2html*, etc).

- Exemple :

```
=head1 NAME
Personne - classe implémentant des personnes
(...)
=head1 SYNOPSIS
$mon_age=$moi->{dire_age};
=head1 DESCRIPTION
(...)
```

Agrégation (1)

Il existe deux façons de réutiliser des objets :

1. L'agrégation.
2. L'héritage.

- On utilise l'agrégation lorsque l'on a un objet *composé* d'autres objets.

- On utilise l'héritage lorsqu'un objet *est un* autre objet (relation de spécialisation).

Agrégation (2)

Imaginons que nous souhaitons qu'une personne puisse avoir un surnom (un surnom étant un objet) :

```
package Personne;

sub new {
  my($classe,$un_nom,$un_age,$un_surnom) = @_;
  my $self = {};
  $self->{NOM} = $un_nom;
  $self->{AGE} = $un_age;
  $self->{PRENOMS} = [];
  $self->{SURNOM} = Surnom->new($un_surnom);
  bless($self,$classe);
  return($self);
}
```

Héritage (1)

- L'héritage permet à un objet d'*étendre* un autre objet (on dit aussi spécialiser).

- Lorsqu'une classe hérite d'une autre, elle recupère toutes ses méthodes et attributs.

- En Perl, la (ou les) classe(s) mères sont stockées dans le tableau @ISA (variable globale - déclarée non pas via *my* mais *our*).

Héritage (2)

Imaginons que nous avons la classe Employe héritant de Personne (l'employé est une personne ayant un statut) :

```
package Employe;
use Personne;

our @ISA = ("Personne");
sub est_embauche {
  my($classe,$un_nom,$un_age,$un_statut) = @_;
  my $self = $classe->SUPER::new($un_nom,$un_age);
  $self->{STATUT} = $un_statut;
  bless($self,$classe);
  return($self);
}
```

Héritage (3)

Notons enfin, qu'il est possible d'utiliser une méthode de classe en tant que méthode d'objet, en utilisant la fonction `ref` :

```
package Personne;

sub new {
  my($classe,$un_nom,$un_age) = @_;
  $classe = ref($classe) || $classe ;
  my $self = {};
  $self->{NOM} = $un_nom;
  $self->{AGE} = $un_age;
  $self->{PRENOMS} = [];
  bless($self,$classe);
  return($self);
}
```

L'intérêt de cela est de prévenir tout problème en cas d'oubli du SUPER

dans la définition de `est_embauche`.

Héritage (4)

L'appel à la fonction `ref` permet de tester si le 1er argument du constructeur est une référence ou non.

- si oui elle nous renvoie la classe de l'objet correspondant à la référence,
- sinon, via le ||, l'instruction nous donne directement la classe appelant la méthode.

Par ce procédé, il est possible d'utiliser des méthodes de classe comme des méthodes d'objets, c'est-à-dire que les deux appels suivants sont tolérés :

```
my $a = Personne->new();
my $b = $a->new();
```

Remarque

Lors de l'écriture d'un programme orienté objet en Perl, nous avons les éléments suivants :

- un fichier contenant la définition d'une classe (nommé *maclasse.pm*), débutant par l'instruction `package maclasse;`
- un fichier *client* manipulant des objets de *maclasse*, contenant l'instruction `use maclasse;`

Exemple de script *client* :

```
#!/usr/bin/perl
use strict;
use Personne;

my $pers = Personne->new("Jean",21);
$pers->nommer("Pierre");
print "Mon age est ".$pers->dire_age()." ans\n";
```