Perl et la programmation orientée objet

De la base à la modernité

par djibril

Date de publication : 9 juillet 2009

Dernière mise à jour : 21 mars 2012

Le but de cet article est de vous exposer la façon la plus simple et la plus classique de faire de la Programmation Orientée Objet en Perl, puis de vous montrer les dernières recommandations de la communauté Perl.

A - Brève introduction de Perl et de la Programmation Orientée Objet....3
A-1 - Avantages....3
A-2 - Inconvénients....3
B - Les bases de la POO en Perl....3
B-1 - Les classes et les objets....3
B-1-a - Définition d'une classe....4
B-1-b - Définition d'un constructeur....4
B-1-c - Les objets....6
B-1-d - Amélioration du constructeur et de la création de l'objet....7
B-1-e - Remarques sur les constructeurs, objets et attributs....7
B-2 - Les méthodes....9
B-2-a - Les méthodes d'instance....9
B-2-b - Les méthodes de classes ou statiques....12
B-3 - Bon à savoir : la protection des méthodes....14
B-4 - Le destructeur....15
B-5 - L'encapsulation....16
B-6 - L'héritage....16
B-6-a - L'héritage simple....17
B-6-b - L'héritage multiple....21
B-7 - Le polymorphisme....24
B-8 - Modules et méthodes utiles....24
B-8-a - Version d'une classe....24
B-8-b - Classe UNIVERSAL....24
B-8-c - Méthode AUTOLOAD....26
B-8-d - Module Class::ISA....26
B-8-e - Autres modules....26
B-9 - Modèles, classes complètes et scripts....27
C - La modernité de la POO en Perl....36
C-1 - Moose - la modernité....36
C-2 - Moose - Les inconvénients....43
C-3 - Coat - le petit dernier....43
D - Les techniques avancées, les objets inversés....44
D-1 - Notion d'objet inversé....44
D-2 - Class::Std::Utils....44
D-3 - Class::InsideOut....46
D-4 - Object::InsideOut....50
D-5 - Comparaison entre ces trois modules....56
E - Conclusion....57
F - Références utilisées....57
G - Remerciements....57

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

A - Brève introduction de Perl et de la Programmation Orientée Objet

La **P**rogrammation **O**rientée **O**bjet (qu'on notera **POO**) est un concept possédant de nombreuses vertus universellement reconnues à ce jour. C'est une méthode de programmation qui permet d'améliorer le développement et la maintenance d'applications, de logiciels avec un gain de temps non négligeable. Il est important de garder à l'esprit qu'elle ne renie pas la programmation structurée (ou procédurale) puisqu'elle est fondée sur elle. L'approche classique pour introduire la POO dans un langage existant est de passer par une encapsulation des fonctionnalités existantes. Le langage C++ s'est construit sur le C et a apporté la POO. Le langage JAVA est fondé sur la syntaxe du C++.

Perl a également suivi le pas en proposant des extensions pour donner la possibilité à ses fans de pouvoir utiliser ce paradigme de programmation. Néanmoins, Perl étant permissif, il n'est pas aussi strict que des langages "pur objet". Mais c'est normal, car la philosophie de Perl est maintenue, *"there is more than one way to do it"* (**TIMTOWTDI**). Dans les sections **A** et **B** nous allons vous expliquer comment fonctionne la POO en Perl traditionnellement, néanmoins en Perl moderne, certains modules facilitent grandement et rendent bien plus élégante la POO, nous vous les présenterons en section **C**.

A-1 - Avantages

Il est plus facile et rapide de modifier, de faire évoluer, de maintenir du code issu d'un logiciel, d'une application de moyenne ou de grande envergure avec la POO.
L'architecture du code peut permettre à d'autres applications de réutiliser des composants, des classes...
D'autres développeurs peuvent facilement utiliser vos programmes. C'est le cas des modules du CPAN que l'on a l'habitude d'utiliser : ils sont tous écrits en POO ! Je tiens à tout de même préciser que les modules simples sont aussi réutilisables. La clarté d'un module n'est pas liée au paradigme de programmation.

A-2 - Inconvénients

La philosophie de Perl peut être un inconvénient à la POO car il existe des dizaines de façons différentes d'écrire un programme, des modules...
Sachez que la POO dégrade en général les performances et qu'il est important de savoir si ce paradigme de programmation est adapté à votre problématique. La POO nécessite sans doute une plus grande réflexion quant à la définition de l'architecture du programme, avant de passer à la programmation (ce qui peut être un avantage). Dans tous les cas, comme le disait **Damian Conway** :

Utilisez la POO pour ses avantages et malgré ses inconvénients et pas simplement parce que c'est le gros marteau familier et confortable qui figure dans votre boite à outils.

B - Les bases de la POO en Perl

B-1 - Les classes et les objets

Lorsque l'on débute en POO, les notions de **classe** et d'**objet** peuvent nous paraître abstraites. Une mauvaise compréhension peut être fatale pour la suite de cet article. Voici une explication de *Sylvain Lhuiller* que je trouve simple et concise.

- **Explication de la POO en Perl par Sylvain Lhuiller**

La programmation orientée objet est un type de programmation qui se concentre principalement sur les données. La question qui se pose en POO est « quelles sont les données du problème ? » par opposition à la programmation procédurale par exemple, qui pose la question « quelles sont les fonctions/actions à faire ? ». En POO, on parle ainsi d'objets,

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

*auxquels on peut affecter des **variables/attributs** (**propriétés**) et des **fonctions/actions** (**méthodes**).*
*On parle de **« classe »**, qui est une manière de représenter des données et comporte des traitements : une classe **« Chaussure »** décrit, par exemple, les caractéristiques d'une chaussure. Elle contient un champ décrivant la pointure, la couleur, la matière, etc. Une telle classe comporte de plus des traitements sur ces données ; ces traitements sont appelés **« méthodes »**. Grossièrement une méthode est une fonction appliquée à un objet. Exemples de méthode : monter, coudre, ressemeler...*
Une fois définie une telle classe, il est possible d'en construire des instances : une instance de classe est dite être un objet de cette classe. Dans notre exemple, il s'agirait d'une chaussure dont la pointure, la couleur et la matière sont renseignées.

B-1-a - Définition d'une classe

Commençons les choses sérieuses en parlant Perl! Vous verrez qu'il est simple de parler Perl avec l'accent orienté objet !
Définir une classe est très simple car ce n'est rien d'autre qu'un package, un module en Perl. Un objet n'est autre chose qu'une référence à un scalaire, un hachage, un tableau (voire une fonction, un typeglob...) qui est liée à cette classe.
Voici notre classe **"Personne"** :

Personne.pm
```perl
package Personne;     # Nom du package, de notre classe
use warnings;         # Avertissement des messages d'erreurs
use strict;           # Vérification des déclarations
use Carp;             # Utile pour émettre certains avertissements
# ...
# ...

1;                  # Important, à ne pas oublier
__END__             # Le compilateur ne lira pas les lignes après elle
```

Voilà! notre classe est créée, simple non !! Il suffit de créer un package du nom de la classe dans un fichier .pm du même nom.
Notre script principal appelle la classe via un **use** habituel.

ScriptPrincipal.pl
```perl
use Personne;
```

B-1-b - Définition d'un constructeur

Pour pouvoir créer un objet et instancier la classe **"Personne"**, on doit définir un **constructeur** dans la classe, c'est-à-dire le fichier *Personne.pm* (sachez qu'un script ".pm" peut avoir plusieurs classes, mais dans cet article, on travaillera avec une classe par fichier ".pm").
Créer un constructeur Perl revient à créer une fonction spéciale nommée **"new"**. Le constructeur peut être nommé **"create"**, **"forge"** ou tout autre nom. Et il peut exister plusieurs constructeurs. Perl n'impose pas grand-chose sur le constructeur comme dans les autres langages OO. Néanmoins, c'est le nom standard utilisé en Perl POO. Dans un souci de maintenance et de clarté, pourquoi déroger à la règle !! Sachez seulement que ce n'est pas un opérateur comme dans certains langages POO.

Personne.pm - constructeur
```perl
sub new {
  # ...
}
```

L'appel du constructeur dans le script principal se fait de la façon suivante :

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.
http://djibril.developpez.com/tutoriels/perl/poo/

ScriptPrincipal.pl
```
my $Personne = Personne->new();
```

On reviendra plus tard sur cet appel.
Notre classe **"Personne"** est caractérisée par son nom, son prénom, son âge, son sexe et son nombre d'enfants. Ces informations sont transmises par l'utilisateur de notre classe (le script principal) au constructeur. Ce dernier s'attend donc à 5 arguments. Mais en fait non ! Ce sera 6 car le premier argument correspond toujours au nom de la classe. Mais l'utilisateur de la classe n'a pas à se soucier de cela, il ne passe que 5 arguments au constructeur. Créons 5 champs (attributs), attributs pour cette classe (*nom, prénom, âge, sexe, nombre d'enfants*).

Personne.pm - constructeur
```
sub new {
  my ( $classe, $nom, $prenom, $age, $sexe, $nombre_enfant ) = @_;
  # ...
}
```

Tout constructeur doit déterminer si son premier argument est le nom de la classe ou une référence :

Personne.pm - constructeur : vérification de la classe
```
$classe = ref($classe) || $classe;
```

Si **$classe** est une référence (ce qui signifie que c'est une instance d'une classe, on en parlera plus loin dans l'article), on prend ref($classe) (c'est-à-dire que le constructeur a été appelé à partir d'un objet *$djibril->new();*), sinon, $classe est la chaîne contenant la classe (c'est-à-dire que le constructeur a été appelé à partir du nom de la classe : *Personne->new();*). *ref* retourne le nom de la classe lorsque la référence passée en paramètre est une référence à un objet (et non une référence simple, non blessed).
Vous pouvez d'ores et déjà tester pour vous amuser.

Personne.pm - constructeur : vérification de la classe
```
$classe = ref($classe) || $classe;
print $classe;
```

Script principal
```
#!/usr/bin/perl
use warnings;
use strict;
use Carp;

use Personne;

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

Cela a pour effet d'afficher la chaîne *Personne*.
Créons maintenant une référence anonyme à un hachage vide qui stockera les attributs de notre classe et deviendra l'objet de la classe. Nous la nommons **$this**.

Personne.pm - constructeur : futur objet
```
my $this = {};
```

Si vous avez un souci de compréhension des références Perl, notre FAQ **FAQ Perl** est votre amie.
Nous avons choisi de nommer notre variable « $this » par analogie aux autres langages OO, mais ce n'est pas obligatoire. D'ailleurs, si vous consultez le code source de certains modules Perl du CPAN, vous verrez qu'elles sont souvent nommées **$self**.
Lions notre variable $this à notre classe :

Personne.pm - constructeur : liaison de la référence à notre classe
```
bless($this, $classe);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

A ce stade, notre objet est créé et lié à la classe **"Personne"**. Si vous l'affichez (*print $this*), vous obtiendrez ceci ***Personne=HASH(0x235bb0)*** alors qu'avant la *bénédiction* (bless), on obtenait ***HASH(0x235bb0)***. L'adresse mémoire correspondant au hachage est liée à la classe.

Vous ne dormez pas encore j'espère !!
Stockons maintenant les attributs de notre classe que l'utilisateur passera via le constructeur.

Personne.pm - constructeur : arguments

```
$this->{_NOM}           = $nom;
$this->{_PRENOM}        = $prenom;
$this->{AGE}            = $age;
$this->{_SEXE}          = $sexe;
$this->{NOMBRE_ENFANT} = $nombre_enfant;
```

Voilà, fini !! Plus qu'à retourner l'objet à l'utilisateur !

Personne.pm - constructeur : retourner l'objet

```
return $this;
```

Vous aurez remarqué que les attributs d'un constructeur (noms des clés) sont en **majuscules**. De plus, certaines clés sont **précédées d'un souligné "_"**, cela signifie que les valeurs des clés ne devraient pas être modifiables par l'utilisateur. Ce ne sont que des conventions, mais je vous les recommande.
Dans notre exemple, une personne ne peut pas changer de nom, de prénom et de sexe !!
Voilà à quoi ressemble notre premier constructeur !

Personne.pm - constructeur

```
sub new {
  my ( $classe, $nom, $prenom, $age, $sexe, $nombre_enfant ) = @_;

  # Vérifions la classe
  $classe = ref($classe) || $classe;

  # Création de la référence anonyme de hachage vide (futur objet)
  my $this = {};

  # Liaison de l'objet à la classe
  bless $this, $classe;

  $this->{_NOM}           = $nom;
  $this->{_PRENOM}        = $prenom;
  $this->{AGE}            = $age;
  $this->{_SEXE}          = $sexe;
  $this->{NOMBRE_ENFANT} = $nombre_enfant;

  return $this;
}
```

B-1-c - Les objets

Avec ce peu de code, on peut déjà instancier notre classe en créant des objets depuis notre script principal.

Script principal - Objet

```
#!/usr/bin/perl
use warnings;
use strict;
use Carp;

use Personne;

my $Objet_Personne1 = Personne->new('Dupont', 'Jean', 45, 'M', 3);
```

Nous venons de créer notre objet, on pourrait en créer plusieurs.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Remarquez bien la notation : la classe est suivie d'une flèche qui pointe sur le constructeur **"new"** auquel on passe des arguments. Il existe une autre notation correcte, mais pouvant porter à confusion.

Script principal - Objet
```
my $Objet_Personne1 = new Personne('Dupont', 'Jean', 45, 'M', 3);
```

Cela donne l'impression que **new** est un opérateur spécifique de Perl et que l'on appelle une méthode **Personne** à laquelle on passe des arguments. Ce qui est bien sûr faux ! Donc gardez en tête la notation recommandée avec flèche.

B-1-d - Amélioration du constructeur et de la création de l'objet

Avant de continuer notre apprentissage de Perl et la POO, prenons de bonnes habitudes. Notre constructeur a un petit inconvénient : Il attend 5 arguments. Vous me direz oui et alors !!! Il est dangereux en termes de maintenance et de lisibilité d'écrire le code comme énoncé ci-dessus. En effet, on ne sait jamais dans quel ordre passer les arguments, il est facile de se tromper. De plus, si l'on souhaite ne pas préciser l'âge de notre personne, il faudrait penser à passer un **undef**. La bonne manière consiste donc à passer une référence anonyme à un hachage avec en clé des variables claires et compréhensibles et en valeur la donnée. Voici ce que l'on peut obtenir :

Personne.pm - constructeur
```
sub new {
  my ( $classe, $ref_arguments ) = @_;

  # Vérifions la classe
  $classe = ref($classe) || $classe;

  # Création de la référence anonyme d'un hachage vide (futur objet)
  my $this = {};

  # Liaison de l'objet à la classe
  bless( $this, $classe );

  $this->{_NOM}           = $ref_arguments->{nom};
  $this->{_PRENOM}        = $ref_arguments->{prenom};
  $this->{AGE}            = $ref_arguments->{age};
  $this->{_SEXE}          = $ref_arguments->{sexe};
  $this->{NOMBRE_ENFANT} = $ref_arguments->{nombre_enfant};

  return $this;
}
```

Script principal
```
my $Objet_Personne1 = Personne->new(
  { nom           => 'Dupont',
    prenom        => 'Jean',
    age           => 45,
    sexe          => 'M',
    nombre_enfant => 3,
  }
);
```

C'est quand même beaucoup plus propre et évolutif non ?

B-1-e - Remarques sur les constructeurs, objets et attributs

- Il est possible d'avoir plusieurs constructeurs dans une classe. Ces derniers seront conçus de la même manière que le constructeur new. Il faudra cependant penser à changer de nom !
- Un objet peut être un autre type de référence que celle d'un hachage.
- L'utilisateur peut voir le contenu de l'objet (puisque c'est un hachage), il suffit d'utiliser le module **Data::Dumper**.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Script principal
```
use Data::Dumper;
print Dumper $Objet_Personne1;
```

Résultat - On obtient
```
$VAR1 = bless(
  { '_PRENOM'        => 'Jean',
    'AGE'            => 45,
    '_SEXE'          => 'M',
    'NOMBRE_ENFANT' => 3,
    '_NOM'           => 'Dupont'
  },
  'Personne'
);
```

- L'utilisateur peut donc modifier le contenu

Il peut modifier les valeurs des attributs de la classe puisqu'ils ne sont pas protégés ou en créer de nouveaux.

Script principal
```
use Data::Dumper;
my $Objet_Personne1 = Personne->new(
  { nom           => 'Dupont',
    prenom        => 'Jean',
    age           => 45,
    sexe          => 'M',
    nombre_enfant => 3,
  }
);
print Dumper $Objet_Personne1;

# modification de l'objet
$Objet_Personne1->{_NOM}     = 'toto';
$Objet_Personne1->{MECHANT} = 'developpeur';
print "===== modification =====\n";
print Dumper $Objet_Personne1;
```

Résultat
```
$VAR1 = bless( {
                 '_PRENOM' => 'Jean',
                 'AGE' => 45,
                 '_SEXE' => 'M',
                 'NOMBRE_ENFANT' => 3,
                 '_NOM' => 'Dupont'
               }, 'Personne' );
===== modification =====
$VAR1 = bless( {
                 'MECHANT' => 'developpeur',
                 '_PRENOM' => 'Jean',
                 'AGE' => 45,
                 '_SEXE' => 'M',
                 'NOMBRE_ENFANT' => 3,
                 '_NOM' => 'toto'
               }, 'Personne' );
```

Perl fait confiance à ses bons développeurs. Donc ces derniers n'iront pas jouer avec les attributs de la classe ! Lorsque vous utilisez une voiture, vous ne vous préoccupez pas de savoir comment elle a été conçue, de connaître le rôle d'une vis lambda du moteur. Vous n'utilisez que ce qui vous a été mentionné dans la notice. De toute façon, pourquoi prendre le risque de modifier le comportement d'une classe qui fonctionne bien ? Vous pouvez toujours vous amuser à modifier les pièces du moteur de votre voiture dans votre jardin au risque de ne plus pouvoir rouler avec !! Sinon, il est généralement de bonne pratique de créer une méthode permettant d'obtenir la valeur d'un attribut ou de la modifier. On les appelle des **accesseurs** ou **mutateurs**.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

B-2 - Les méthodes

Une méthode n'est autre chose qu'une fonction en Perl. La seule différence est qu'elle a toujours en premier argument l'objet créé par le constructeur. On parle de **méthode d'instance**. Si cet argument est plutôt le nom de la classe (**$classe**), on parle de **méthode de classe**.

B-2-a - Les méthodes d'instance

Une méthode d'instance est une fonction Perl qui s'applique à une instance de la classe (à l'objet). Elle a toujours pour premier argument l'objet ($this). Pour illustrer cela, créons deux méthodes dans la classe **"Personne"**. On donne la possibilité à une personne de pouvoir marcher et parler, logique !!

Personne.pm - méthode marcher et parler

```
# Méthode marcher - ne prend aucun argument
sub marcher {
  my $this = shift;

  print "[$this->{_NOM} $this->{_PRENOM}] marche\n";

  return;
}

# Méthode parler - un argument
sub parler {
  my ( $this, $message ) = @_;

  unless ( defined $message ) {
    $message = 'ne sais pas quoi dire';
  }

  # Le premier caractère de la phrase est mis en majuscule
  $message = ucfirst $message;
  print "[$this->{_NOM} $this->{_PRENOM}] $message\n";

  return 1;
}
```

La méthode **marcher** fait marcher un objet personne et n'a pas besoin d'arguments. Par contre, pour parler, il faut dire quelque chose même si ce n'est pas obligatoire ! Pour appeler ces méthodes, c'est simple :

Script principal

```
$Objet_Personne1->marcher();
$Objet_Personne1->parler('Bonjour tout le monde');
$Objet_Personne1->parler();
```

Résultat

```
[Dupont Jean] Marche
[Dupont Jean] Bonjour tout le monde
[Dupont Jean] Ne sais pas quoi dire
```

Les **accesseurs** et les **mutateurs** sont des méthodes qui permettent à l'utilisateur de la classe de lire ou modifier la valeur d'un attribut proprement.
Supposons que l'on souhaite modifier la valeur du champ **age**. Le mauvais programmeur va modifier directement le contenu de l'objet en faisant ceci :

Script principal

```
#!/usr/bin/perl
use warnings;
use strict;
use Carp;

use Personne;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Script principal

```
my $Objet_Personne1 = Personne->new( {
  nom     => 'Dupont',
  prenom   => 'Jean',
  age     => 45,
  sexe     => 'M',
  nombre_enfant => 3,
});

print $Objet_Personne1->{AGE},"\n";
$Objet_Personne1->{AGE} = 22;
print $Objet_Personne1->{AGE},"\n";
```

```
45
22
```

Cette pratique n'est pas conseillée car on viole les attributs de la classe. Mais comme les perléens codent proprement, ils liront la documentation de la classe et trouveront les méthodes adéquates que voici :

Personne.pm - accesseur, mutateur âge

```
# accesseur obtenir_age
sub obtenir_age {
  my ( $this ) = @_;
  return $this->{AGE};
}

# mutateur modifier_age
sub modifier_age {
  my ( $this, $age ) = @_;

  if ( defined $age ) {
    $this->{AGE} = $age;
  }

  return;
}
```

Ces deux méthodes permettent d'obtenir (obtenir_age) et modifier (modifier_age) l'âge d'une personne. La méthode **modifier_age** attend un argument *$age* afin de modifier l'âge de notre personne. Certains développeurs préfèrent avoir une unique méthode du même nom que l'attribut jouant le rôle de mutateur et d'accesseur comme ici :

Personne.pm - accesseur/mutateur age

```
# accesseur/mutateur age
sub age {
  my ( $this, $age ) = @_;

  if ( defined $age ) {
    $this->{AGE} = $age;
  }

  return $this->{AGE};
}
```

Cette méthode **age** retourne l'âge de la personne si aucun argument ne lui est transmis, sinon elle le modifie. Il est vrai que cette méthode est simple, qu'il y a moins de codes à maintenir, mais il est tout de même préférable d'avoir un mutateur et accesseur par attribut pour ces quelques raisons :

- au premier regard, c'est beaucoup plus lisible ;
- il y a peu de chances de confondre une méthode quelconque avec une méthode ayant le rôle de mutateur/ accesseur ;
- il est inutile de faire des tests à chaque appel d'un accesseur, on retourne juste la valeur.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

De plus, si l'on fait appel à un mutateur **$objet->age($age);** avec **$age = undef;**, Perl ne signalera aucune erreur car la méthode **age** se comportera comme un accesseur. Il est alors très difficile de trouver d'où vient l'erreur dans un tel programme. Pourtant, confondre **obtenir_age** et **modifier_age** n'est pas dramatique, car il est plus facile de retrouver l'erreur. Voilà !!

Script principal

```
print "Mon age : ",$Objet_Personne1->obtenir_age(),"\n";
$Objet_Personne1->modifier_age(22);
print "Mon nouvel age : ",$Objet_Personne1->obtenir_age(),"\n";
```

résultat

```
45
Mon nouvel age : 22
```

C'est quand même beaucoup plus propre ! Vous pouvez maintenant créer deux méthodes par attribut si vous jugez nécessaire que l'utilisateur puisse le lire et le modifier.
Petit résumé, voici à quoi ressemble notre classe **"Personne"** actuellement :

Personne.pm

```
package Personne;     # Nom du package, de notre classe
use warnings;         # Avertissement des messages d'erreurs
use strict;           # Vérification des déclarations
use Carp;             # Utile pour émettre certains avertissements

# Constructeur de la classe Personne
sub new {
  my ( $classe, $ref_arguments ) = @_;

  # Vérifions la classe
  $classe = ref($classe) || $classe;

  # Création de la référence anonyme à un hachage vide (futur objet)
  my $this = {};

  # Liaison de l'objet à la classe
  bless( $this, $classe );

  $this->{_NOM}           = $ref_arguments->{nom};
  $this->{_PRENOM}        = $ref_arguments->{prenom};
  $this->{AGE}            = $ref_arguments->{age};
  $this->{_SEXE}          = $ref_arguments->{sexe};
  $this->{NOMBRE_ENFANT} = $ref_arguments->{nombre_enfant};

  return $this;
}

# accesseur obtenir_nom
sub obtenir_nom {
  my $this = shift;
  return $this->{_NOM};
}

# accesseur obtenir_prenom
sub obtenir_prenom {
  my $this = shift;
  return $this->{_PRENOM};
}

# accesseur obtenir_sexe
sub obtenir_sexe {
  my $this = shift;
  return $this->{_SEXE};
}

# accesseur obtenir_age
sub obtenir_age {
  my $this = shift;
  return $this->{AGE};
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Personne.pm

```
}

# mutateur modifier_age
sub modifier_age {
  my ( $this, $age ) = @_;

  if ( defined $age ) {
    $this->{AGE} = $age;
  }

  return;
}

# accesseur obtenir_nombre_enfant
sub obtenir_nombre_enfant {
  my $this = shift;
  return $this->{NOMBRE_ENFANT};
}

# mutateur modifier_nombre_enfant
sub modifier_nombre_enfant {
  my ( $this, $nombre_enfant ) = @_;

  if ( defined $nombre_enfant ) {
    $this->{NOMBRE_ENFANT} = $nombre_enfant;
  }

  return;
}

# Méthode marcher - ne prend aucun argument
sub marcher {
  my $this = shift;

  print "[$this->{_NOM} $this->{_PRENOM}] Marche\n";

  return;
}

# Méthode parler - un argument
sub parler {
  my ( $this, $message ) = @_;

  unless ( defined $message ) {
    $message = 'Ne sais pas quoi dire';
  }

  # Le premier caractère de la phrase est mis en majuscule
  $message = ucfirst($message);
  print "[$this->{_NOM} $this->{_PRENOM}] $message\n";

  return 1;
}

1;    # Important, à ne pas oublier
__END__           # Le compilateur ne lira pas les lignes suivantes
```

B-2-b - Les méthodes de classes ou statiques

Une méthode de classe s'applique à toute une classe et non à l'instance de la classe (l'objet $this). Une variable statique est une variable commune à toute une classe, exemple :

Personne.pm - champs statiques

```
package Personne;    # Nom du package, de notre classe
use warnings;        # Avertissement des messages d'erreurs
use strict;          # Vérification des déclarations
use Carp;            # Utile pour émettre certains avertissements
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Personne.pm - champs statiques

```
# Comptage du nombre de personnes créées
my $NbrPersonnes = 0;         # champ statique privé
our $VariablePublique = 12;   # champ statique public

sub new {
# ...
}

# ...

1;
__END__
```

Les deux variables **$NbrPersonnes** et **$VariablePublique** sont statiques car accessibles à tout moment dans la classe. Mais comme il se trouve qu'elles ont aussi une portée globale à tout le package, elles sont accessibles partout et à tout moment dans le module. **$VariablePublique** déclarée avec **our** est publique car elle peut être accessible en dehors de la classe, du module. **$NbrPersonnes** déclarée avec **my** est privée, sa portée se limite à la classe **"Personne"**, au module.

Script principal

```
#!/usr/bin/perl
use warnings;
use strict;
use Carp;

use Personne;

print "NbrPersonnes ",$Personne::NbrPersonnes,"\n";
print "VariablePublique ",$Personne::VariablePublique,"\n";
```

Résultat

```
Name "Personne::NbrPersonnes" used only once: possible typo at ...
Use of uninitialized value in print ...
NbrPersonnes
VariablePublique 12
```

Pour lire depuis le script principal une variable statique $NbrPersonnes qui est déclarée dans la classe **"Personne"**, il faut écrire **$Personne::NbrPersonnes** (**$***NomDeLaClasse***::*****NomVariable***). Dans la classe "Personne", on a déclaré **$VariablePublique** avec **our**, elle est donc publique. Par contre, **$NbrPersonnes** est privée (à cause de my) et sa portée s'arrête à la classe. C'est pour cette raison que dans notre exemple ci-dessus, on n'arrive pas à afficher le contenu de $NbrPersonnes.

Une méthode de classe n'a accès qu'aux variables statiques et non à l'objet $this. Elle a la particularité de prendre en premier argument le nom de la classe (comme le constructeur).

Pour mieux comprendre, créons une méthode nommée **"Obtenir_nbr_personnes"** dont le but sera de nous retourner le nombre de personnes créées dans notre script. Dans le constructeur new, on incrémentera la variable statique **$NbrPersonnes**.

Personne.pm - méthode statique Obtenir_nbr_personnes

```
# Comptage du nombre de personnes créées
my $NbrPersonnes = 0;         # champ statique privé

# Constructeur de la classe Personne
sub new {
  my ( $classe, $ref_arguments ) = @_;

  # Vérifions la classe
  $classe = ref($classe) || $classe;

  # Création de la référence anonyme à un hachage vide (futur objet)
  my $this = {};

  # Liaison de l'objet à la classe
  bless( $this, $classe );
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Personne.pm - méthode statique Obtenir_nbr_personnes

```
  $this->{_NOM}           = $ref_arguments->{nom};
  $this->{_PRENOM}        = $ref_arguments->{prenom};
  $this->{AGE}            = $ref_arguments->{age};
  $this->{_SEXE}          = $ref_arguments->{sexe};
  $this->{NOMBRE_ENFANT} = $ref_arguments->{nombre_enfant};

  # Nombre de personnes créées.
  $NbrPersonnes++;

  return $this;
}

# méthode de classe Obtenir_nbr_personnes
sub Obtenir_nbr_personnes {
  my $classe = shift;

  return $NbrPersonnes;
}
```

Pour appeler cette méthode dans notre script principal, la notation sera légèrement différente.

Script principal

```
#!/usr/bin/perl
use warnings;
use strict;
use Carp;

use Personne;

my $Objet_Personne1 = Personne->new(
  { nom           => 'Dupont',
    prenom        => 'Jean',
    age           => 45,
    sexe          => 'M',
    nombre_enfant => 3,
  }
);

print "Nbr personne : ",Personne->Obtenir_nbr_personnes(),"\n";

my $Objet_Personne2 = Personne->new(
  { nom           => 'Durand',
    prenom        => 'Djibril',
    age           => 35,
    sexe          => 'M',
    nombre_enfant => 0,
  }
);
print "Nbr personne : ",Personne->Obtenir_nbr_personnes(),"\n";
```

Résultat

```
Nbr personne : 1
Nbr personne : 2
```

La flèche est placée sur le nom de la classe et non sur l'objet.
N.B. En écrivant **$Personne1->Obtenir_nbr_personnes();**, on aurait eu le même résultat car dans notre méthode, on n'utilise pas l'argument **$classe**, mais si c'était le cas, nous aurions eu quelques soucis.

B-3 - Bon à savoir : la protection des méthodes

Il est possible de protéger les méthodes d'une classe en utilisant des références. En créant une référence à une fonction (méthode) ayant une portée limitée au package via **my**, la méthode devient inaccessible de l'extérieur de la classe.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

méthodes privées
```
my $_private_method = sub {
  my $this = shift;

  my @argument = @_;
  # ...
}
```

Pour utiliser cette méthode au sein de la classe, utilisons la référence.

```
# Appel de la méthode privée
sub public_method {
  # ...
  $this->$_private_method(...);  # ou $this->$_private_method->(...);
  # ...
}
```

Sachez qu'il existe différentes façons de créer des méthodes à la volée en utilisant les **"fermetures"** en Perl, mais cela sort du cadre de cet article.

B-4 - Le destructeur

Lorsqu'un objet est détruit, Perl libère la mémoire occupée par lui. Un objet **$Personne1** est détruit lorsqu'il est égal à **undef** ou à la fin de sa portée (ex : s'il a été créé dans un bloc (for, if, else...), il sera détruit à la fin de la boucle).
Le destructeur est une méthode spéciale nommée **"DESTROY"** qui est appelée (si elle est définie dans la classe), lorsqu'un objet est détruit, juste avant la libération de la mémoire de celui-ci. Contrairement au constructeur dont le nom est laissé libre au programmeur, celui du destructeur est forcé par les mécanismes OO de Perl.
Elle peut nous permettre d'effectuer des tâches avant la destruction de l'objet :

- fermer une connexion réseau, se déconnecter d'une base de données, fermer un fichier ;
- afficher un message ;
- appeler le destructeur d'une classe héritée ;
- décrémenter le nombre de personnes créées ;
- etc.

Voici notre destructeur :

Personne.pm - destructeur
```
# Destructeur
sub DESTROY {
  my $this = shift;

  # Avertissement que l'objet va mourir
  print "===================================================\n";
  print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
  print "===================================================\n";

  # Diminuer le nombre de personnes créées.
  $NbrPersonnes--;

  return;
}
```

Comme les méthodes d'instance, son premier argument doit être l'objet **$this** et c'est tout. Perl se charge de l'appeler et du reste !
__PACKAGE__ est une variable de paquetage Perl qui contient le nom de la classe.
Prenons un exemple :

Script principal
```
#!/usr/bin/perl
use warnings;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Script principal

```
use strict;
use Carp;

use Personne;

my $Objet_Personne1 = Personne->new(
  { nom           => 'Dupont',
    prenom        => 'Jean',
    age           => 45,
    sexe          => 'M',
    nombre_enfant => 3,
  }
);

print "Bonjour\n";
$Objet_Personne1 = undef;
print "Bye bye";
```

Résultat

```
Bonjour
===================================================
L'objet de la classe Personne va mourir
===================================================
Bye bye
```

Quand on exécute ce script, Perl crée un objet **$Objet_Personne1**, affiche *Bonjour*, détruit l'objet, et affiche *Bye bye*. Nous remarquons que **DESTROY** est appelée au bon moment.

B-5 - L'encapsulation

L'encapsulation est un mécanisme consistant à rassembler les données, les attributs et les méthodes au sein d'une classe en cachant l'implémentation de l'objet. C'est un concept permettant de protéger l'intégrité des données d'un objet et de donner la possibilité à l'utilisateur de lire ou modifier les valeurs des attributs via des méthodes accesseurs, mutateurs.
Pour être plus précis, l'utilisateur ne doit utiliser que ce qu'on lui permet. S'il n'y a pas de mutateurs, il ne doit pas pouvoir modifier les attributs. Il ne doit pouvoir utiliser que les méthodes publiques. Le concepteur d'une classe se doit de protéger le reste.
Lorsque vous utilisez une voiture, vous ne vous préoccupez pas de savoir comment elle a été conçue, de connaître de rôle d'une vise lambda du moteur. Vous n'utilisez que ce qui vous a été mentionné dans la notice. Cela n'a pas empêché les constructeurs des voitures de protéger l'accès à certaines parties du moteur afin d'éviter que des malins fassent n'importe quoi. Je sais, c'est un peu tiré par les cheveux, mais faut bien trouver un exemple parlant.
Certains langages de programmation comme JAVA, C++, C# utilisent des mots-clés comme **public**, **private**, **protected** pour protéger des méthodes, des classes. Mais comme vous le savez, Perl est *permissif*. Il fait confiance à ses programmeurs de bonne éducation ! Une des conventions à respecter est que toute méthode ou nom d'attribut précédé d'un souligné **"_"** est privé (**Ex : sub _methode {}**).
Si vous ne faites pas trop confiance à vos confrères programmeurs, il est possible de créer des attributs et méthodes vraiment privés. Cette notion de protection sera abordée dans la **section D** de cet article.

B-6 - L'héritage

Le concept d'héritage constitue l'un des fondements de la POO. Il permet la réutilisation des classes, donc d'hériter des fonctionnalités (attributs, méthodes). Perl permet de bénéficier de ce concept.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

B-6-a - L'héritage simple

Pour l'illustrer, créons deux nouvelles classes **"Homme"** et **"Femme"**. Un homme ou une femme est une personne, donc a un nom, un prénom, un âge, un sexe, et un nombre d'enfants. Un homme peut être barbu ou non. Ces classes héritent de la classe **"Personne"**, donc de ses attributs et de ses méthodes.

- **Lien de filiation**

Signalons le lien de filiation entre classes au moyen du module **base**.

```
package Homme;
use base qw( Personne ); # Hérite de la classe Personne
```

Cette ligne suffit pour que la classe **"Homme"** hérite de la classe **"Personne"**. Dans d'autres documentations, il est mentionné d'utiliser le tableau **@ISA**.

```
our @ISA = qw ( Personne );
```

L'inconvénient d'utiliser cette variable de paquetage est que les hiérarchies des classes sont établies par des affectations à la phase d'exécution. En utilisant le module **base**, on définit la hiérarchie d'une classe avec une déclaration à la phase de compilation. On s'assure ainsi que l'héritage est établi dès que possible et de charger automatiquement le module *Personne.pm*.

- **Le constructeur**

La conception d'un constructeur d'une classe héritant d'une autre est légèrement différente d'une classe n'héritant d'aucune. Rappelez-vous, lorsque nous avons créé notre constructeur **new** dans la classe **"Personne"**, nous avons créé une référence anonyme à un hachage vide. Maintenant, dans la classe **"Homme"**, ce ne sera pas le cas. Notre classe **"Homme"** utilisera l'objet de la classe **"Personne"**.

```
my $this = $classe->SUPER::new($ref_arguments);
```

On utilise ici une pseudoclasse nommée **SUPER** (pour supérieur) qui désigne la classe mère (*Personne*). La variable **$this** du constructeur de la classe **"Homme"** (ou **"Femme"**) est ainsi l'équivalent du *$this* de la classe **"Personne"** et on lui passe les attributs qu'il attend.

- **Les attributs**

Prenons pour exemple la classe **"Homme"**. Son constructeur réceptionne les arguments que le script principal lui transmet (*nom*, *prenom*, *sexe*, *age*, *nombre_enfant* et le nouveau *barbu*). Ensuite, il ne se préoccupe pas de créer l'attribut *_NOM*, *_PRENOM*, *AGE*, *_SEXE* et *NOMBRE_ENFANT* puisque la classe **"Personne"** dont il hérite s'en charge. Pour ce faire, il suffit de transmettre ces arguments à la classe **"Personne"** via la pseudoclasse **SUPER** citée ci-dessus. Nous récupérons ainsi l'objet de la classe **"Personne"** $this. Nous la bénissons afin de la lier à notre classe **"Homme"** (avec bless), puis nous y ajoutons l'attribut *barbu* (le petit nouveau). Maintenant notre objet est prêt à être renvoyé au script principal. Voici le résultat ci-dessous :

Homme.pm - constructeur

```
# Constructeur de la classe Homme
sub new {
  my ( $classe, $ref_arguments ) = @_;

  # Vérifions la classe
  $classe = ref($classe) || $classe;

  # Objet de notre classe héritée
  my $this = $classe->SUPER::new(
    { nom          => $ref_arguments->{nom},
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Homme.pm - constructeur

```
      prenom        => $ref_arguments->{prenom},
      age           => $ref_arguments->{age},
      sexe          => 'M',
      nombre_enfant => $ref_arguments->{nombre_enfant},
    }
  );

  # Liaison de l'objet à la classe
  bless( $this, $classe );

  # Nouvel attribut
  $this->{BARBU} = $ref_arguments->{barbu};

  return $this;
}
```

Le constructeur de la classe **"Femme"** est presque identique à deux lignes près. Le sexe sera **'F'** et il n'y aura pas d'attribut barbu, logique !!
Créons l'accesseur et le mutateur de l'attribut barbu.

Homme.pm - accesseur/mutateur barbu

```
# accesseur obtenir_barbu
sub obtenir_barbu {
  my $this- = shift;
  return $this->{BARBU};
}

# mutateur modifier_barbu
sub modifier_barbu {
  my ( $this, $barbu ) = @_;

  if ( defined $barbu ) {
    $this->{BARBU} = $barbu;
  }

  return $this->{BARBU};
}
```

Nous n'avons pas besoin de créer les mutateurs et accesseurs pour l'âge, le nombre d'enfants, etc. Les classes **"Homme"** et **"Femme"** héritent déjà de ces méthodes. C'est beau l'héritage !!

- **Le destructeur**

Dans le cas de l'héritage, il faut penser à appeler également le destructeur de la classe mère (Personne), car Perl s'arrête au premier destructeur rencontré. C'est-à dire qu'à la mort de notre objet $Objet_Personne1 dans le script principal, Perl appelle le destructeur de la classe **"Homme"** (ou **"Femme"**) puis libère la mémoire. Il n'est pas encore assez intelligent pour penser à appeler le destructeur de la classe dont il hérite ("Personne"), faut tout lui dire !!. Pour ce faire, on recourt à la pseudoclasse **SUPER**.

Homme.pm ou Femme.pm - Destructeur

```
# Destructeur
sub DESTROY {
  my $this = shift;

  # destructeur de la classe mère.
  print "=>Appel du destructeur de la classe mère\n\n";
  $this->SUPER::DESTROY();

  # Avertissement que l'objet va mourir
  print "====================================================\n";
  print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
  print "====================================================\n";

  return;
}
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Faisons un petit test et admirez le résultat.

Script principal
```
#!/usr/bin/perl
use warnings;
use strict;
use Carp;

use Homme;

my $Objet_Personne1 = Homme->new(
  { nom           => 'Dupont',
    prenom        => 'Jean',
    age           => 45,
    nombre_enfant => 3,
  }
);

print "Bonjour\n";
$Objet_Personne1 = undef;
print "Bye bye";
```

Résultat
```
Bonjour
=>Appel du destructeur de la classe mère

===================================================
L'objet de la classe Personne va mourir
===================================================
===================================================
L'objet de la classe Homme va mourir
===================================================
Bye bye
```

- **Les méthodes**

Comme il a été dit ci-dessus, nos classes **"Homme"** et **"Femme"** héritent de la classe **"Personne"**. De ce fait, elles héritent de ses méthodes. Si dans notre script principal, nous souhaitons créer une femme (objet femme) et que nous souhaitons la faire parler, il suffit juste de faire appel à la méthode **parler** comme nous savons déjà le faire :

Script principal
```
use Femme;
my $Femme = Femme->new(
  { nom           => 'Dupont',
    prenom        => 'Anne',
    age           => 45,
    nombre_enfant => 3,
  }
);
$Femme->parler("bonjour");
```

Résultat
```
[Dupont Anne] Bonjour
```

Il n'y a rien d'autre à faire. Notre classe **"Femme"** n'a pas besoin d'avoir une méthode **parler**. En fait, Perl cherche la méthode **parler** dans la classe **"Femme"**, il ne la trouve pas et continue sa recherche dans la classe héritée (Personne). Il la trouve et l'exécute. C'est magique !

- **Redéfinition de méthodes**

Supposons maintenant que l'on souhaite faire parler dans notre script principal un homme, le principe est le même que pour une femme, on appelle la méthode **parler** de la même façon. Pour une raison quelconque, nous décidons qu'un homme doit s'affirmer quand il parle, et la seule façon de le faire est de préciser qu'il est un homme à chaque fois

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

qu'il parle (effet de la testostérone !). Lorsque le script principal lui demande de dire *"bonjour"*, au lieu d'afficher *"[Dupont Jean] Bonjour"* par exemple comme c'était le cas, il affichera *"[Dupont Jean] Bonjour (je suis un homme)"* !!! Pour faire cela, on ne va pas créer une nouvelle méthode s'appelant par exemple **parler_homme**, pourquoi ? Imaginons qu'on ait une centaine de scripts utilisant notre classe **"Homme"**, ça veut dire que pour rendre nos hommes virils, il faudrait ouvrir tous les scripts et changer les méthodes **parler** en **parler_homme** en faisant attention de ne pas en oublier ou de se tromper en faisant parler une femme comme un homme. En terme de maintenance, c'est une catastrophe. Utilisons tout simplement le concept de **redéfinition**.
Créons une méthode **parler** dans notre classe **"Homme"**. On dira qu'on a redéfini la méthode **parler** car cette dernière existe déjà dans la classe mère **"Personne"**. Voici à quoi elle ressemble :

```
# Méthode parler (un argument) qui va redéfinir la méthode parler
# de la classe mère
sub parler {
  my ( $this, $message ) = @_;

  unless ( defined $message ) {
    $message = 'je suis un homme et Ne sais pas quoi dire';
  }
  $this->SUPER::parler($message . ' (je suis un homme)');

  return 1;
}
```

Explication : Si la méthode **parler** est appelée avec un argument (un message $message), on rajoute la phrase *"(je suis un homme)"* sinon, notre message est *"je suis un homme et Ne sais pas quoi dire"*. Ensuite, au lieu faire un print, on fait appel à la méthode **parler** de la classe mère **"Personne"** (autant profiter de l'héritage et ne pas réinventer la roue).

```
use Homme;
my $Homme1 = Homme->new(
  { nom           => 'Dupont',
    prenom        => 'Jean',
    age           => 45,
    nombre_enfant => 3,
  }
);

$Homme1->marcher();
$Homme1->parler('Bonjour');
```

```
[Dupont Jean] Marche
[Dupont Jean] Bonjour (je suis un homme)
```

Créons maintenant une méthode **accouche** dans notre classe **"Femme"** qui hérite évidemment de la classe **"Personne"**, elle ne redéfinit aucune méthode. Elle permet à une femme d'accoucher d'un nombre **x** d'enfant(s) qu'on lui passe en argument.

Femme.pm - méthode accouche

```
# méthode accouche
sub accouche {
  my ( $this, $nombre_enfant ) = @_;

  if ( defined $nombre_enfant ) {
    unless ( $nombre_enfant =~ m{^\d+$} ) {
      croak "Mauvais argument : $nombre_enfant\n";
      return;
    }

    $this->parler("accouche de $nombre_enfant enfant(s)");
    my $NbrEnfant = $this->obtenir_nombre_enfant() + $nombre_enfant;
    $this->modifier_nombre_enfant($NbrEnfant);
  }

  return;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Femme.pm - méthode accouche
```
}
```

Script principal
```
#!/usr/bin/perl
use warnings;
use strict;
use Carp;

use Femme;

my $Femme1 = Femme->new( {
  nom => 'Dupont',  prenom => 'Jeanne',
  age => 45,        barbu => 1,
  nombre_enfant => 3
});


$Femme1->parler("J'ai " . $Femme1->obtenir_nombre_enfant() . " enfant(s)");
$Femme1->accouche(1);
$Femme1->parler("J'ai " . $Femme1->obtenir_nombre_enfant() . " enfant(s)");
```

Résultat
```
[Dupont Jeanne] J'ai 3 enfant(s)
[Dupont Jeanne] Accouche de 1 enfant(s)
[Dupont Jeanne] J'ai 4 enfant(s)
```

C'est super ! Notre méthode **accouche** joue bien son rôle. Elle affiche un message pour signaler son accouchement d'un nombre d'enfants. Ensuite elle met à jour le champ *NOMBRE_ENFANT* en utilisant le mutateur de la classe **"Personne"**. Maintenant vous maîtrisez le mécanisme d'héritage simple. Passons à l'héritage multiple !

B-6-b - L'héritage multiple

Une classe peut hériter des fonctionnalités de plusieurs classes, c'est l'héritage multiple. Pour continuer sur notre lancée, nos hommes et femmes ont besoin de travailler pour subvenir à leurs besoins quotidiens. Par conséquent ils vont payer des impôts pour le bien de notre pays. Nous devons créer une méthode qui sera capable de calculer leurs impôts. Pour ce faire, nous avons besoin de connaître le montant du salaire et pourquoi pas le métier tant qu'à faire. Créons donc une classe nommée **"Activite"** dont les attributs seront *SALAIRE* et *METIER*. Cette classe aura des mutateurs et accesseurs pour les attributs *metier* et *salaire*, puis une méthode **impot_a_payer**, sans oublier notre constructeur. Voici la classe :

Activite.pm
```
package Activite;     # Nom du package, de notre classe
use warnings;         # Avertissement des messages d'erreurs
use strict;           # Vérification des déclarations
use Carp;             # Utile pour émettre certains avertissements

# Constructeur de la classe Personne
sub new {
  my ( $classe, $ref_arguments ) = @_;

  # Vérifions la classe
  $classe = ref($classe) || $classe;

  # Objet de notre classe héritée
  my $this = {};

  # Liaison de l'objet à la classe
  bless( $this, $classe );

  $this->{METIER}  = $ref_arguments->{metier};
  $this->{SALAIRE} = $ref_arguments->{salaire};

  return $this;
}
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Activite.pm

```
# accesseur obtenir_salaire
sub obtenir_salaire {
  my $this = shift;

  return $this->{SALAIRE};
}

# mutateur modifier_salaire
sub modifier_salaire {
  my ( $this, $salaire ) = @_;

  if ( defined $salaire ) {
    $this->{SALAIRE} = $salaire;
  }

  return;
}

# accesseur obtenir_metier
sub obtenir_metier {
  my $this = shift;

  return $this->{METIER};
}

# mutateur modifier_metier
sub modifier_metier {
  my ( $this, $metier ) = @_;

  if ( defined $metier ) {
    $this->{METIER} = $metier;
  }

  return;
}

# méthode impot_a_payer
sub impot_a_payer {
  my $this = shift;

  if ( exists $this->{SALAIRE} ) {
    return 0.6 * $this->{SALAIRE};
  }
  croak("[Attention] Difficile de calculer vos impôts sans salaire\n");
}

# Destructeur
sub DESTROY {
  my $this = shift;

  # Avertissement que l'objet va mourir
  print "===================================================\n";
  print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
  print "===================================================\n";

  return;
}

1;    # Important, à ne pas oublier
__END__            # Le compilateur ne lira pas les lignes suivantes
```

Notre classe est prête. La petite nouveauté pour vous est peut-être la fonction Perl **croak**. C'est une fonction exportée du module **Carp**. C'est l'équivalent d'un **die**. Sauf qu'au lieu de donner le numéro de la ligne du module ".pm" quand elle sera appelée, elle donnera le numéro de la ligne du script où la méthode **impot_a_payer** a été appelée. C'est plus facile pour l'utilisateur de savoir d'où vient le problème.
Bon, c'est bien, mais on ne parle toujours pas d'héritage multiple !!! Nous y voilà. Pour que nos classes **"Homme"** et **"Femme"** puissent utiliser les méthodes de la classe **"Activité"**, héritons-en. Rien de plus simple, une seule ligne est à changer dans les classes **"Homme"** et **"Femme"**.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Femme.pm et Homme.pm
```
use base qw ( Personne Activite ); # hérite de la classe Personne et Activite
```

Au lieu de

Femme.pm et Homme.pm
```
use base qw( Personne ); # Hérite de la classe Personne
```

On peut maintenant utiliser les méthodes **impot_a_payer** et les mutateurs.

Script principal
```
#!/usr/bin/perl
use warnings;
use strict;
use Carp;

use Femme;

my $Femme1 = Femme->new(
  { nom           => 'Dupont',
    prenom        => 'Jeanne',
    age           => 45,
    nombre_enfant => 1,
  }
);

$Femme1->modifier_metier("Informaticien");
$Femme1->modifier_salaire(1500);
$Femme1->parler( "Je vais payer cette année un impôt de : "
    . $Femme1->impot_a_payer()
    . " euros"
    . " pour un boulot "
    . $Femme1->obtenir_metier() );
```

```
[Dupont Jeanne] Je vais payer cette année un impôt de : 900 euros pour un boulot Informaticien
=>Appel du destructeur de la classe mère

===================================================
L'objet de la classe Personne va mourir
===================================================
===================================================
L'objet de la classe Femme va mourir
===================================================
```

Explication : La classe **"Activite"** contient ses accesseurs et une méthode **impot_a_payer**. Notre classe **"Femme"** hérite de la classe **"Personne"** et **"Activite"**. L'ordre est important. Lorsque nous appelons la méthode **impot_a_payer** dans le script principal, Perl va chercher s'il existe dans la classe **"Femme"**, puis dans la classe **"Personne"**, puis continue sa recherche dans les classes dont "Personne" hérite (aucune), puis dans **"Activite"**, etc. Si nous avions une méthode **impot_a_payer** dans la classe Personne, c'est elle qui aurait été appelée.
Remarque : : Supposons que vous êtes dans une classe **"Enfant"** qui hérite des classes **"Personne"** et **"JeuxVideos"**.

```
use base qw ( Personne JeuxVideos );
```

Vous souhaitez utiliser la méthode **parler** de la classe **"JeuxVideos"** qui existe dans les deux classes héritées. Si vous faites :

```
$this->SUPER::parler('blablabla');
```

Perl cherche dans l'ordre et la trouve en premier dans la classe **"Personne"**. Pour contourner ce problème, vous devez l'appeler ainsi :

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

```
$this->JeuxVideos::parler('blablabla');
```

Quelques précautions à prendre.

- L'ordre d'héritage multiple a une importance, faites attention.
- Dans vos classes, soyez sûr des méthodes que vous appelez dans le cas d'héritage multiple.
- Choisissez des noms d'attributs compréhensibles.
- Faites attention à ne pas écraser vos noms d'attributs (clés de hachage) en cas d'héritage.
- Dans vos classes héritant de plusieurs autres, n'oubliez pas d'appeler les destructeurs des classes mères (voir Class::ISA).

N'hésitez pas à tester les valeurs de vos attributs. Vous imaginez bien qu'un homme ne peut pas avoir **-2** enfant(s).
Lors d'un héritage multiple, comme dans notre classe **"Homme"** et **"Femme"**, **SUPER** ne désignant que la première classe de base (classe **"Personne"**), seul le constructeur de cette base peut être appelé, et par conséquent, les méthodes des autres classes sont bien "héritées", mais les propriétés doivent être initialisées grâce aux mutateurs de ces autres classes de base. Pour revenir à nos classes "Homme" et "Femme", elles héritent des classes "Personne" et "Activité". Leurs constructeurs via la pseudoclasse **SUPER** utilisent le constructeur de la classe "Personne". Si nous souhaitons donner le nom d'un métier à notre homme ou notre femme, il faut utiliser le mutateur **modifier_metier** (héritée de la classe **"Activite"**).

B-7 - Le polymorphisme

Le terme de *polymorphisme* indique qu'une entité peut apparaître suivant plusieurs formes. C'est un concept puissant en POO qui complète l'héritage. Il permet en quelque sorte le choix d'une méthode parmi plusieurs de même nom quand la situation se présente. En fait, nous l'avons déjà abordé sans le savoir.
Si vous vous souvenez, dans nos classes **"Personne"** et **"Homme"**, nous avons une méthode **parler**. Dans notre script principal, en fonction que l'on utilise la classe **"Homme"** ou **"Femme"**, la méthode **parler** utilisée n'est pas la même car nous avons redéfini cette méthode dans notre classe **"Homme"**, on peut parler de polymorphisme. Cette possibilité de redéfinir une méthode dans des classes héritant d'une classe de base s'appelle en POO pure **"la spécialisation"**.
Le polymorphisme en Perl se fait tout seul grâce au module **base** ou à la variable **@ISA**. On peut aussi soit redéfinir une méthode, ou bien choisir d'appeler la méthode de la classe héritée que l'on souhaite.
Rappel sur la recherche de méthodes : lorsque vous appelez une méthode sur un objet, l'interpréteur Perl la cherche premièrement dans la classe de l'objet. Puis il monte dans la hiérarchie de classes jusqu'à la trouver. Si la recherche est infructueuse dans l'arbre d'héritage, il revient à la classe la plus dérivée et recommence.

B-8 - Modules et méthodes utiles

B-8-a - Version d'une classe

Il est important de donner une version à votre classe. Cela permet de suivre facilement l'évolution de votre classe :

Classe .pm
```
use vars qw($VERSION);
$VERSION = '1.00'; # Version de notre module
```

ou

Classe .pm
```
our $VERSION = '1.00'; # Version de notre module
```

B-8-b - Classe UNIVERSAL

En Perl, il existe une classe mère, ancêtre de toutes les classes, c'est la classe **UNIVERSAL**. Elle contient trois méthodes **isa**, **can** et **VERSION**.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Quelle est l'utilité de ces méthodes ?

- **isa**

Si nous souhaitons connaître la classe principale d'un objet, on peut utiliser la fonction Perl **ref**. Pour rappel, lorsque vous appelez cette fonction sur une référence, elle vous retourne le type de la structure de données sur laquelle elle pointe (SCALAR, ARRAY, HASH...). De ce fait, **ref($objet);** retournera le nom de la classe.

```
print ref $Femme1; # => Femme
```

Si nous souhaitons savoir si notre objet **$Femme1** est un objet de la classe **"Personne"** (ce qui est le cas via l'héritage), **ref** ne peut pas nous aider. Il faut recourir à la méthode **isa** de la classe **UNIVERSAL**.

```
print "\$Femme1 appartient objet de la classe Femme\n"    if ( $Femme1->isa('Femme') );
print "\$Femme1 appartient objet de la classe Activite\n" if ( $Femme1->isa('Activite') );
print "\$Femme1 appartient objet de la classe Personne\n" if ( $Femme1->isa('Personne') );
print "\$Femme1 appartient objet de la classe Homme\n"    if ( $Femme1->isa('Homme') );
```

```
$Femme1 appartient objet de la classe Femme
$Femme1 appartient objet de la classe Activite
$Femme1 appartient objet de la classe Personne
```

On constate qu'**isa** nous indique que notre objet **$Femme1** n'appartient pas à la classe **"Homme"**, encore heureux, mais qu'il appartient bien aux autres.

- **can**

La méthode **can** renvoie la référence d'une méthode appelée si elle existe, sinon, elle nous retourne **undef**.

```
print $Femme1->can('parler'); # CODE(0x187df20)
print $Femme1->can('toto'); # use of uninitilized value in print ...
```

- **VERSION**

La méthode **VERSION** renvoie la version de notre classe si cette dernière a été mentionnée. Elle permet :

- d'obtenir la version de notre classe. Si elle n'en a pas, un message d'erreur est retourné ;
- de vérifier que la version de la classe est plus grande qu'un réel qu'on lui passerait en argument.

Supposons que notre programme principal utilise la classe **"Femme"**. Nous souhaitons vérifier que la version de la classe **"Femme"** est récente et au moins supérieure à la version *4.3* dans le cas contraire, notre script ne fonctionnera pas. On va appeler la méthode **VERSION** et lui passer en paramètre *4.3*.

```
use Femme;
print 'Version : ',Femme->VERSION();
Femme->VERSION('4.3');
```

Cette ligne suffit à faire la vérification. Si la version de la classe **"Femme"** est inférieure à *4.3*, on obtient le message suivant :

Mauvaise version

```
Femme version 4.3 required--this is only version 1.07 at ...
Version : 1.0
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

B-8-c - Méthode AUTOLOAD

La méthode **AUTOLOAD** est appelée automatiquement par l'interpréteur Perl lorsqu'il ne trouve pas une méthode d'une classe. Lorsque vous appelez une méthode sur un objet, l'interpréteur Perl la cherche premièrement dans la classe de l'objet. Puis il monte dans la hiérarchie de classes jusqu'à la trouver. Si la recherche est infructueuse dans l'arbre d'héritage, il revient à la classe la plus dérivée et recommence. Dans cette seconde recherche, il cherche la méthode **AUTOLOAD**.
Cette méthode **ne favorise pas** l'efficacité, la concision, la robustesse et la maintenabilité. **Il faut l'éviter complètement** pour plusieurs raisons que je ne citerai pas ici.
En voici une en provenance du livre **Perl Best Practices** de *Damian Conway* :
L'AUTOLOAD, "le-plus-à-gauche-en-recherche-en-profondeur" dont l'objet hérite sera appelé systématiquement pour traiter tous les appels à une méthode inconnue. C'est un problème. Si la hiérarchie de la classe de l'objet comporte deux définitions AUTOLOAD() ou plus, il se peut que la seconde soit plus à même de traiter tel ou tel appel de méthode inconnue. Mais normalement, ce deuxième sous-programme n'aura jamais la possibilité de le faire.

B-8-d - Module Class::ISA

Ce module permet de lister toutes les classes mères d'une classe. Il retourne la hiérarchie des classes dérivées. Il fournit trois méthodes qui ne sont pas exportables. Pour les appeler, il faut écrire :

1. Class::ISA::**super_path**($Classe) : retourne un tableau contenant les classes mères dans l'ordre de recherche, $Classe n'est pas inclus dans le tableau ;
2. Class::ISA::**self_and_super_path**($Classe) : retourne un tableau contenant les classes mères dans l'ordre de recherche, $Classe est inclus dans le tableau et se trouve en première position ;
3. Class::ISA::**self_and_super_versions**($Classe) : retourne un hachage contenant en clés les classes mères et en valeurs, les versions.

Ces méthodes peuvent être utiles pour l'appel des destructeurs des classes mères dans l'héritage simple et multiple.

B-8-e - Autres modules

Il existe plusieurs modules sur le CPAN qui permettent de faire de la POO en Perl différemment. Ils peuvent nous fournir des mécanismes pour mieux protéger nos attributs, méthodes, etc. Ces modules sont pour la plupart nommés **Class::***, **Object::***, nous parlerons de certains de ces modules plus tard dans cet article.

Module Carp

- Utilisez la méthode **croak** de ce module au lieu d'utiliser *die*
- Utilisez la méthode **carp** au lieu d'utiliser la fonction *warn*

```
use Carp;
croak('A la place de die');
carp('A la place de warn');
```

croak() et **carp()** servent à se plaindre d'un mauvais usage de votre classe et signale donc l'erreur dans le script appelant, s'il s'agit d'un dysfonctionnement interne, il faut utiliser **die()** et **warn()**.
Dans notre constructeur, nous vérifions que l'utilisateur passe en argument les attributs *nom*, *prenom* et *sexe*. S'il en oublie, on stoppe le script (car cela sera fatal pour notre script) en lui précisant la ligne d'erreur dans son script avec **croak**.

```
# Vérification des champs nom, prenom et sexe
  foreach my $attribut (qw/ nom prenom sexe /) {
    unless ( defined $ref_arguments->{$attribut} ) {
      croak("[Attention] Attribut $attribut manquant");
    }
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

```
    }
```

Il obtiendra cette erreur :

```
[Attention] Attribut prenom manquant
 at C:\PATH\PerlPOOBases\ScriptPrincipal.pl line 14
```

Il pourra facilement faire la correction en ligne 14 de son script. Alors qu'un **die** aurait provoqué cette erreur :

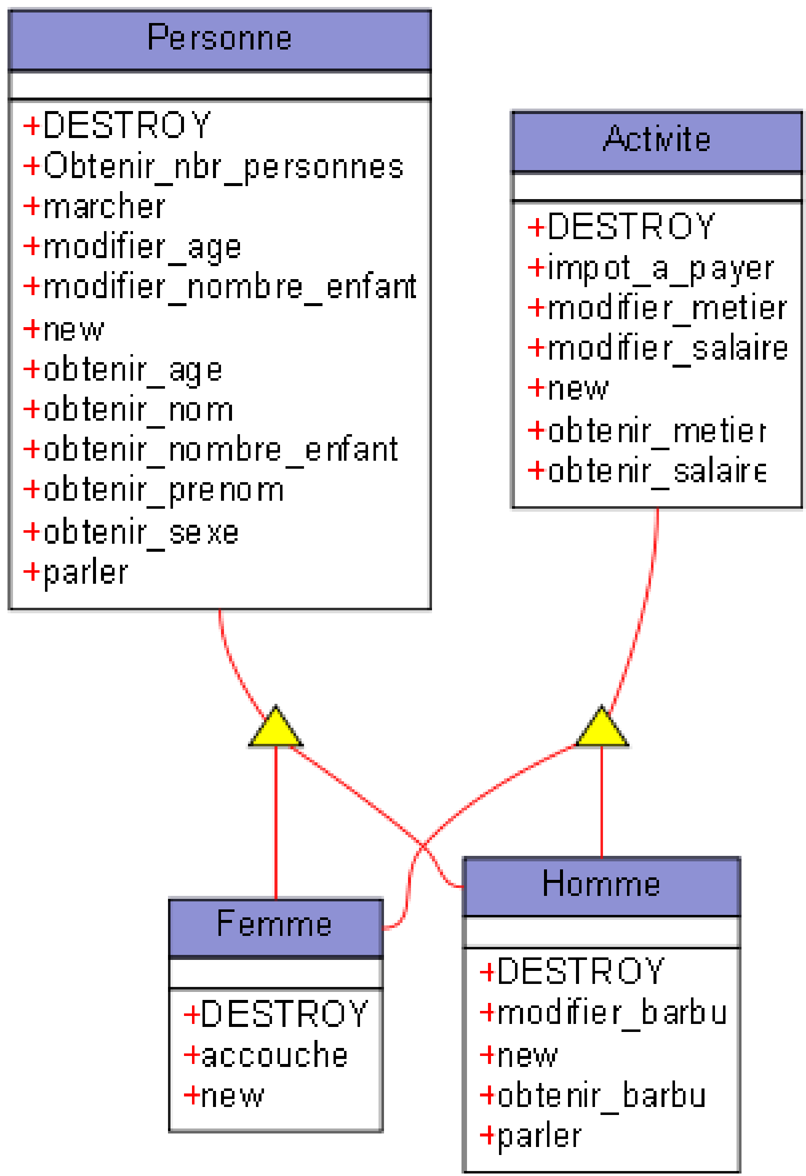
```
[Attention] Attribut prenom manquant at Personne.pm line 28.
```

Ce qui n'aide pas vraiment l'utilisateur.

B-9 - Modèles, classes complètes et scripts

Modèle de nos classes et méthodes
Cette image a été créée à partir du module **UML::Class::Simple** qui permet de créer une image représentant toutes les classes d'un module, d'un répertoire, etc. Pour l'utiliser, il est nécessaire d'installer **Graphviz**. L'image nous montre que les classes **"Homme"** et **"Femme"** héritent des classes **"Personne"** et **"Activite"**. Pour chaque classe, les méthodes publiques sont affichées.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.



Exemple de modèle par rapport à nos classes

Code complet de toutes nos classes et script principal les utilisant

Le but du script est de faire parler six personnes afin de jouer avec toutes les méthodes que nous avons créées.

Vous pouvez les télécharger **ICI**.

Classe Personne.pm
```
package Personne;     # Nom du package, de notre classe
use warnings;         # Avertissement des messages d'erreurs
use strict;           # Vérification des déclarations
use Carp;             # Utile pour émettre certains avertissements

use vars qw($VERSION);     # Version de notre module
$VERSION = '1.00';

# Comptage du nombre de personnes créées
my $NbrPersonnes = 0;      # champ statique privé

# Constructeur de la classe Personne
sub new {
  my ( $classe, $ref_arguments ) = @_;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe Personne.pm

```
  # Vérifions la classe
  $classe = ref($classe) || $classe;

  # Création de la référence anonyme à un hachage vide (futur objet)
  my $this = {};

  # Liaison de l'objet à la classe
  bless( $this, $classe );

  # Vérification des champs nom, prenom et sexe
  foreach my $attribut (qw/ nom prenom sexe /) {
    unless ( defined $ref_arguments->{$attribut} ) {
      croak("[Attention] Attribut $attribut manquant\n");
    }
  }
  unless ( $ref_arguments->{sexe} =~ m{^M|F$}i ) {
    croak(
      "[Attention] Mauvais attribut sexe : $ref_arguments->{sexe}, F ou M\n");
  }

  $this->{_NOM}           = $ref_arguments->{nom};
  $this->{_PRENOM}        = $ref_arguments->{prenom};
  $this->{AGE}            = $ref_arguments->{age};
  $this->{_SEXE}          = $ref_arguments->{sexe};
  $this->{NOMBRE_ENFANT} = $ref_arguments->{nombre_enfant} || 0;

  # Nombre de personnes créées.
  $NbrPersonnes++;

  return $this;
}

# accesseur obtenir_nom
sub obtenir_nom {
  my $this = shift;
  return $this->{_NOM};
}

# accesseur obtenir_prenom
sub obtenir_prenom {
  my $this = shift;
  return $this->{_PRENOM};
}

# accesseur obtenir_sexe
sub obtenir_sexe {
  my $this = shift;
  return $this->{_SEXE};
}

# accesseur obtenir_age
sub obtenir_age {
  my $this = shift;
  return $this->{AGE};
}

# mutateur modifier_age
sub modifier_age {
  my ( $this, $age ) = @_;

  if ( defined $age ) {
    unless ( $age =~ m{^\d+$} ) {
      croak "Mauvais argument : $age\n";
      return;
    }
    $this->{AGE} = $age;
  }

  return;
}
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe Personne.pm

```
# accesseur obtenir_nombre_enfant
sub obtenir_nombre_enfant {
  my $this = shift;

  return $this->{NOMBRE_ENFANT};
}

# mutateur modifier_nombre_enfant
sub modifier_nombre_enfant {
  my ( $this, $nombre_enfant ) = @_;

  if ( defined $nombre_enfant ) {
    unless ( $nombre_enfant =~ m{^\d+$} ) {
      croak "Mauvais argument : $nombre_enfant\n";
      return;
    }
    $this->{NOMBRE_ENFANT} = $nombre_enfant;
  }

  return;
}

# Méthode marcher - ne prend aucun argument
sub marcher {
  my $this = shift;

  print "[$this->{_NOM} $this->{_PRENOM}] Marche\n";

  return;
}

# Méthode parler - un argument
sub parler {
  my ( $this, $message ) = @_;

  unless ( defined $message ) {
    $message = 'Ne sais pas quoi dire';
  }

  # Le premier caractère de la phrase est mis en majuscule
  $message = ucfirst($message);
  print "[$this->{_NOM} $this->{_PRENOM}] $message\n";

  return 1;
}

# méthode de classe Obtenir_nbr_personnes
sub Obtenir_nbr_personnes {
  my $classe = shift;

  return $NbrPersonnes;
}

# Destructeur
sub DESTROY {
  my $this = shift;

  # Avertissement que l'objet va mourir
  print "===================================================\n";
  print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
  print "===================================================\n";

  # Diminuer le nombre de personnes créées.
  $NbrPersonnes--;

  return;
}

1;    # Important, à ne pas oublier
__END__           # Le compilateur ne lira pas les lignes suivantes
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe Homme.pm

```
package Homme;     # Nom du package, de notre classe
use warnings;      # Avertissement des messages d'erreurs
use strict;        # Vérification des déclarations
use Carp;          # Utile pour émettre certains avertissements

use base qw ( Personne Activite );    # hérite de la classe Personne et Activite
use vars qw($VERSION);                # Version de notre module
$VERSION = '1.00';

# Constructeur de la classe Personne
sub new {
  my ( $classe, $ref_arguments ) = @_;

  # Vérifions la classe
  $classe = ref($classe) || $classe;

  # Objet de notre classe héritée
  my $this = $classe->SUPER::new(
    { nom           => $ref_arguments->{nom},
      prenom        => $ref_arguments->{prenom},
      age           => $ref_arguments->{age},
      sexe          => 'M',
      nombre_enfant => $ref_arguments->{nombre_enfant},
    }
  );

  # Liaison de l'objet à la classe
  bless( $this, $classe );

  # Nouvel attribut
  $this->{BARBU} = $ref_arguments->{barbu};

  return $this;
}

# accesseur obtenir_barbu
sub obtenir_barbu {
  my $this = shift;

  return $this->{BARBU};
}

# mutateur modifier_barbu
sub modifier_barbu {
  my ( $this, $barbu ) = @_;

  if ( defined $barbu ) {
    unless ( $barbu =~ m{^\d+$} ) {
      croak "Mauvais argument : $barbu\n";
      return;
    }
    $this->{BARBU} = $barbu;
  }

  return $this->{BARBU};
}

# Méthode parler (un argument) qui va redéfinir la méthode parler
# de la classe mère
sub parler {
  my ( $this, $message ) = @_;

  unless ( defined $message ) {
    $message = 'je suis un homme et Ne sais pas quoi dire';
  }
  $this->SUPER::parler( $message . ' (je suis un homme)' );

  return 1;
}

# Destructeur
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe Homme.pm

```
sub DESTROY {
  my $this = shift;

  # destructeur de la classe mère.
  print "=>Appel du destructeur de la classe mère\n\n";
  $this->SUPER::DESTROY();

  # Avertissement que l'objet va mourir
  print "=================================================\n";
  print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
  print "=================================================\n";

  return;
}

1;    # Important, à ne pas oublier
__END__           # Le compilateur ne lira pas les lignes suivantes
```

Classe Femme.pm

```
package Femme;     # Nom du package, de notre classe
use warnings;      # Avertissement des messages d'erreurs
use strict;        # Vérification des déclarations
use Carp;          # Utile pour émettre certains avertissements

use base qw ( Personne Activite );   # hérite de la classe Personne et Activite
use vars qw($VERSION);               # Version de notre module
$VERSION = '1.00';

# Constructeur de la classe Personne
sub new {
  my ( $classe, $ref_arguments ) = @_;

  # Vérifions la classe
  $classe = ref($classe) || $classe;

  # Objet de notre classe héritée
  my $this = $classe->SUPER::new(
    { nom           => $ref_arguments->{nom},
      prenom        => $ref_arguments->{prenom},
      age           => $ref_arguments->{age},
      sexe          => 'F',
      nombre_enfant => $ref_arguments->{nombre_enfant},
    }
  );

  # Liaison de l'objet à la classe
  bless( $this, $classe );

  return $this;
}

# méthode accouche
sub accouche {
  my ( $this, $nombre_enfant ) = @_;

  if ( defined $nombre_enfant ) {
    unless ( $nombre_enfant =~ m{^\d+$} ) {
      croak "Mauvais argument : $nombre_enfant\n";
      return;
    }

    $this->parler("accouche de $nombre_enfant enfant(s)");
    my $NbrEnfant = $this->obtenir_nombre_enfant() + $nombre_enfant;
    $this->modifier_nombre_enfant($NbrEnfant);
  }

  return;
}

# Destructeur
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe Femme.pm

```
sub DESTROY {
  my $this = shift;

  # destructeur de la classe mère.
  print "=>Appel du destructeur de la classe mère\n\n";
  $this->SUPER::DESTROY();

  # Avertissement que l'objet va mourir
  print "===================================================\n";
  print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
  print "===================================================\n";

  return;
}
1;    # Important, à ne pas oublier
__END__          # Le compilateur ne lira pas les lignes suivantes
```

Classe Activite.pm

```
package Activite;    # Nom du package, de notre classe
use warnings;        # Avertissement des messages d'erreurs
use strict;          # Vérification des déclarations
use Carp;            # Utile pour émettre certains avertissements

use vars qw($VERSION);     # Version de notre module
$VERSION = '1.00';

# Constructeur de la classe Personne
sub new {
  my ( $classe, $ref_arguments ) = @_;

  # Vérifions la classe
  $classe = ref($classe) || $classe;

  # Objet de notre classe héritée
  my $this = {};

  # Liaison de l'objet à la classe
  bless( $this, $classe );

  $this->{METIER}  = $ref_arguments->{metier};
  $this->{SALAIRE} = $ref_arguments->{salaire};

  return $this;
}

# accesseur obtenir_salaire
sub obtenir_salaire {
  my $this = shift;

  return $this->{SALAIRE};
}

# mutateur modifier_salaire
sub modifier_salaire {
  my ( $this, $salaire ) = @_;

  if ( defined $salaire ) {
    $this->{SALAIRE} = $salaire;
  }

  return;
}

# accesseur obtenir_metier
sub obtenir_metier {
  my $this = shift;

  return $this->{METIER};
}
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe Activite.pm

```
# mutateur modifier_metier
sub modifier_metier {
  my ( $this, $metier ) = @_;

  if ( defined $metier ) {
    $this->{METIER} = $metier;
  }

  return;
}

# méthode impot_a_payer
sub impot_a_payer {
  my $this = shift;

  if ( exists $this->{SALAIRE} ) {
    return 0.6 * $this->{SALAIRE};
  }
  croak("[Attention] Difficile de calculer vos impôts sans salaire\n");
}

# Destructeur
sub DESTROY {
  my $this = shift;

  # Avertissement que l'objet va mourir
  print "===================================================\n";
  print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
  print "===================================================\n";

  return;
}

1;    # Important, à ne pas oublier
__END__            # Le compilateur ne lira pas les lignes suivantes
```

Classe ScriptPrincipal.pl

```
#!/usr/bin/perl
use warnings;
use strict;
use Carp;

use Femme;
use Homme;
use utf8;    # On a des accents dans le script

# Nous sommes dans une boulangerie
print "[A la boulangerie]\n\n";

# 1ere personne = enfant
my $enfant = Homme->new(
  { nom    => 'Henry',
    prenom => 'Antoine',
    age    => 5,
  }
);

$enfant->parler("bonjour");

# 2eme personne = boulangere
my $boulangere = Femme->new(
  { nom    => 'Laplanche',
    prenom => 'Caroline',
    age    => 35,
  }
);

$boulangere->parler("bonjour");
$enfant->parler("comment tu t'appelles ?");
$boulangere->parler(
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe ScriptPrincipal.pl

```
  " Je m'appelle " . $boulangere->obtenir_prenom() . " et toi ?" );
$enfant->parler( "Je m'appelle "
    . $enfant->obtenir_prenom()
    . $enfant->obtenir_nom() );
$enfant->parler("t'as quel âge ?");
$boulangere->parler( "J'ai " . $boulangere->obtenir_age() . " ans" );
$enfant->parler("pain mame s'il vous plait");

print "\n[Toujours dans la boulangerie, 2 vieux copains "
  . "d'école primaire se rencontrent]\n\n";

my $copain1 = Homme->new(
  { nom    => 'Henry',
    prenom => 'Guillaume',
    age    => 35,
  }
);

$copain1->parler("comment ça va ?");

my $copain2 = Homme->new(
  { nom    => 'Petit',
    prenom => 'Stephane',
    barbu  => 1,
  }
);

#

$copain2->parler("bien");

my $Copain2Barbu = $copain2->obtenir_barbu();
if ( defined $Copain2Barbu and $Copain2Barbu == 1 ) {
  $copain1->parler("et toi ? tu es barbu maintenant ?");
  $copain2->parler("moi ? barbu ? non juste fatigué...");
}

$copain2->parler("ma femme vient d'accoucher, alors les nuits sont courtes");
$copain2->parler("et toi ? tu as combien d'enfants ?");

$copain1->modifier_nombre_enfant(2);
$copain1->parler( "moi j'en ai " . $copain1->obtenir_nombre_enfant() );
$copain2->parler("quel âge ?");
$copain1->parler( "le dernier a 1 an et commence à marcher. le premier "
    . "commence à parler correctement. d'ailleurs "
    . "c'est lui qui demande du pain à la boulangerie" );

$copain2->parler("tu fais quoi comme métier ?");
$copain1->modifier_metier("ingénieur en informatique");
$copain1->parler( "je suis " . $copain1->obtenir_metier() );
$copain2->parler("Pardon, j'ai pas bien entendu");
$copain1->parler( $copain1->obtenir_metier() );

$enfant->parler("papa, pas de pièce pour le pain");
$copain1->parler("va demander à maman, elle est juste devant la porte");
$enfant->marcher();

print "\n[Mais maman se prend une prune par la police municipale] \n\n";
my $FemmeCopain1 = Femme->new(
  { nom    => 'Henry',
    prenom => 'Anne',
  }
);

$FemmeCopain1->parler( "Quoi ? une contravention pour 5 min de stationnement ? "
    . "Et vous êtes payé combien pour ça ?" );

my $contractuel = Homme->new(
  { nom    => 'Letellier',
    prenom => 'Carlos',
    barbu  => 1,
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe ScriptPrincipal.pl
```
  }

);

$contractuel->parler(
  "1200 par mois, et ce n'est pas encore assez pour me faire insulter ! Vous payez combien d'impôts ?"
);

$FemmeCopain1->modifier_salaire(2000);
$FemmeCopain1->parler(
  "plus de " . $FemmeCopain1->impot_a_payer() . " euros par mois" );
$contractuel->parler(
  "Et bien vous pourrez mettre un visage sur le prochain chèque que vous aurez à régler!"
);

if ( $FemmeCopain1->obtenir_sexe() eq 'M' ) {
  $contractuel->parler("Bonne journée monsieur!");
}
else {
  $contractuel->parler("Bonne journée madame!");
}

$FemmeCopain1->parler( $enfant->obtenir_prenom() . ', '
    . $copain1->obtenir_prenom()
    . " : Allez on rentre à la maison!!" );

print "\n[Conversation terminée]\n";
print "Il y avait ", $boulangere->Obtenir_nbr_personnes(),
  " personne(s) dans la conversation\n\n";
```

C - La modernité de la POO en Perl

A ce stade de l'article, vous maîtrisez les bases de la programmation orientée objet en Perl, les notions de classe, de méthode, de constructeur, la construction d'objets, etc. Nous allons voir des nouveautés qui à ce jour nous facilitent grandement la vie. En fait, certains développeurs de la communauté Perl en avaient assez de devoir à chaque fois réécrire les mêmes codes à savoir, un constructeur, des accesseurs, mutateurs, de l'adapter à Perl 6, etc. Ils trouvaient cela ennuyeux et fastidieux. C'est alors qu'un certain **Stevan Little** a décidé d'écrire un module qui permet d'écrire moins de code car tout est géré par le module. Il devient encore plus facile de faire de la POO et la lisibilité du code est améliorée. Ce module se nomme **Moose**. A ce jour, il est recommandé par la communauté Perl pour la POO. Il est puissant, bien pensé et s'appuie sur le modèle objet de Perl 6 (mais ça reste du Perl 5).

Si vous rencontrez des difficultés pour installer ce module, consultez cet **article**, et si ça ne vous aide pas, le forum Perl est là pour vous.

C-1 - Moose - la modernité

Nous allons recréer nos classes de façon moderne avec le module **Moose** qui est stable et déjà bien utilisé en production dans plusieurs entreprises.
Pour vous montrer comment fonctionne ce module, reprenons les exemples utilisés dans la première partie de ce tutoriel. Commençons par créer notre classe **"Personne"**.

Classe Personne
```
package Personne;     # Nom du package, de notre classe
use Carp;             # Utile pour émettre certains avertissements

use vars qw($VERSION);     # Version de notre module
$VERSION = '2.0';

use Moose;               # Il charge automatiquement strict et warnings

# Tout le code ici

1;
__END__
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Moose charge automatiquement les modules **strict** et **warnings**, il n'est donc plus nécessaire de les charger. Créons maintenant les attributs (*nom*, *prenom*, *age*, *sexe* et *nombre_enfant*), ainsi que les accesseurs et mutateurs. Le nom, le prénom et le sexe sont non modifiables, par contre, l'utilisateur peut consulter et modifier l'âge et le nombre d'enfants. Le nom, le prénom et le sexe de notre personne doivent obligatoirement être mentionnés. Par la même occasion, on vérifiera que l'âge et le nombre d'enfants sont des entiers, etc. Vous verrez qu'il est possible de faire tout ça en peu de lignes de code.

Personne.pm - attributs

```
# Comptage du nombre de personnes créées
my $NbrPersonnes = 0;     # champ statique privé

# Création des attributs
has nom => (
  is       => 'ro',       # Attribut est lisible et non modifiable
  isa      => 'Str',      # Valeur de l'attribut de type chaîne de caractères
  required => 1,          # Attribut obligatoire
  reader => 'obtenir_nom',              # Nom de l'accesseur
  trigger => sub { $NbrPersonnes++; },   # Incrémente à chaque nouvelle personne
);

has prenom => (
  is       => 'ro',
  isa      => 'Str',
  required => 1,
  reader   => 'obtenir_prenom',
);

has sexe => (
  is       => 'ro',
  isa      => 'Str',
  required => 1,
  reader   => 'obtenir_sexe',
  trigger  => sub {
    my ( $this, $sexe ) = @_;
    unless ( $sexe eq 'M' or $sexe eq 'F' ) {
      die("[Attention] Mauvais attribut sexe $sexe : F ou M\n");
    }
  },
);

has age => (
  is       => 'rw',
  isa      => 'Int',
  required => 0,
  reader   => 'obtenir_age',
  writer   => 'modifier_age',
  trigger  => sub {
    my ( $this, $age ) = @_;
    unless ( $age > 0 ) {
      die("[Attention] Mauvais attribut age : $age\n");
    }
  },
);

has nombre_enfant => (
  is       => 'rw',
  isa      => 'Int',                     # Valeur de type entier
  required => 0,
  reader   => 'obtenir_nombre_enfant',
  writer   => 'modifier_nombre_enfant',    # Nom du mutateur
  default  => 0,                         # Valeur par défaut de l'attribut
  trigger  => sub {
    my ( $this, $nombre_enfant ) = @_;
    unless ( $nombre_enfant >= 0 ) {
      die("[Attention] Mauvais attribut nombre_enfant : $nombre_enfant\n");
    }
  },
);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Bon, j'espère qu'au premier coup d'oeil, vous avez compris. Si non, voici quelques explications :
has est une fonction de Moose permettant de créer les attributs. Son premier argument (avant la flèche **=>**) est le nom de l'attribut. Après la flèche, ce sont les options. Il existe beaucoup d'options, n'hésitez pas à consulter la documentation (**Moose::Meta::Attribute** et **Moose**). En ce qui concerne la classe **"Personne"**, les attributs *nom*, *prenom* et *sexe* sont en lecture seule. Il sera donc impossible à l'utilisateur de modifier ces trois champs une fois définis dans le constructeur *new*. Les autres attributs sont **rw** (read-write), donc Moose nous met à disposition un accesseur et un mutateur. Par défaut, une méthode **age** (par exemple) est automatiquement créée et joue le rôle d'accesseur et de mutateur, mais comme il est plus judicieux de les séparer, Moose nous le permet. Il suffit de spécifier les options **reader** et **writer**. Moose crée ainsi automatiquement les méthodes avec ces noms et la méthode **age** n'existe plus. L'option **required** est aussi très intéressante car elle permet de spécifier les attributs qu'il est obligatoire de passer au constructeur de la classe. Une dernière option que nous avons utilisée est **trigger**. C'est ce que Moose appelle un **déclencheur**. Il peuvent s'exécuter avant, pendant ou après l'accès ou la modification des attributs. Dans notre cas, il s'exécute à la création de l'attribut.
Vous pouvez constater que le code est très simple, propre et plus compréhensible.
N.B. Nous n'avons pas besoin de créer un constructeur, Moose le fait pour nous.
Voilà, à ce stade, les attributs et notre constructeur sont créés, il ne reste plus qu'à créer nos différentes méthodes.

Personne.pm - méthodes

```perl
# Méthode marcher - ne prend aucun argument
sub marcher {
  my $this = shift;
  my ($nom, $prenom) =( $this->obtenir_nom, $this->obtenir_prenom);
  print "[$nom $prenom] Marche\n";

  return;
}

# Méthode parler - un argument
sub parler {
  my ( $this, $message ) = @_;

  unless ( defined $message ) {
    $message = 'Ne sais pas quoi dire';
  }

  # Le premier caractère de la phrase est mis en majuscule
  $message = ucfirst($message);
  my ($nom, $prenom) =( $this->obtenir_nom, $this->obtenir_prenom);
  print "[$nom $prenom] $message\n";

  return 1;
}

# méthode de classe Obtenir_nbr_personnes
sub Obtenir_nbr_personnes {
  my $classe = shift;

  return $NbrPersonnes;
}
```

Voilà, rien de nouveau à ce niveau. Notez qu'au lieu d'écrire :

```perl
my ($nom, $prenom) =( $this->obtenir_nom, $this->obtenir_prenom);
```

Il était possible d'écrire :

```perl
my ($nom, $prenom) =( $this->{nom}, $this->{prenom} );
```

Moose stocke ses informations dans une référence de hachage avec en clé, le nom de l'attribut défini grâce à la fonction **has**.
En ce qui concerne le destructeur, Moose nous le crée automatiquement. Néanmoins, si nous avons besoin d'un constructeur pour effectuer des tâches précises, il faut utiliser la méthode **DEMOLISH** fournie par Moose et y mettre notre code. Moose l'exécutera avant la méthode **DESTROY**.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Personne.pm - Destructeur

```
# Destructeur
sub DEMOLISH {
  my $this = shift;

  # Avertissement que l'objet va mourir
  print "===================================================\n";
  print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
  print "===================================================\n";

  # Diminuer le nombre de personnes créées.
  $NbrPersonnes--;

  return;
}
```

Notre classe **"Personne"** est terminée. Créons maintenant les classes **"Homme"**, **"Femme"** et **"Activite"**. Commençons par la classe **"Activite"**.

Activite.pm

```
package Activite;     # Nom du package, de notre classe
use Carp;             # Utile pour émettre certains avertissements

use vars qw($VERSION);     # Version de notre module
$VERSION = '1.30';

use Moose;             # Il charge automatiquement strict et warnings

has salaire => (
  is       => 'rw',
  isa      => 'Int',
  required => 0,
  reader   => 'obtenir_salaire',
  writer   => 'modifier_salaire',
);

has metier => (
  is       => 'rw',
  isa      => 'Str',
  required => 0,
  reader   => 'obtenir_metier',
  writer   => 'modifier_metier',
);

# méthode impot_a_payer
sub impot_a_payer {
  my $this = shift;

  if ( my $salaire = $this->obtenir_salaire ) {
    return 0.6 * $salaire;
  }
  croak("[Attention] Difficile de calculer vos impôts sans salaire\n");
}

# Destructeur
sub DEMOLISH {
  my $this = shift;

  # Avertissement que l'objet va mourir
  print "===================================================\n";
  print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
  print "===================================================\n";

  return;
}

1;    # Important, à ne pas oublier
__END__           # Le compilateur ne lira pas les lignes suivantes
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Passons maintenant aux classes **"Homme"** et **"Femme"**, elles vont nous permettre de parler d'héritage. Ces classes vont hériter des classes **"Personne"** et **"Activite"**. Pour le mentionner à Moose, c'est aussi simple que le module **base**.

Classe Homme ou Femme - héritage
```
use Moose;
extends qw ( Personne Activite );    # hérite de la classe Personne et Activite
```

Maintenant, en ce qui concerne les attributs, Il n'y a absolument rien à faire. Nos classes héritent des attributs des classes mères, donc Moose le gère pour nous, c'est super ! Néanmoins, une femme est de sexe féminin et un homme de sexe masculin ! Il serait stupide de demander aux utilisateurs de préciser le sexe s'ils utilisaient l'une des deux classes. Il va donc falloir le définir à notre classe mère **"Personne"**. Grâce à la fonction **has** de Moose, il est possible de cloner l'attribut sexe de **"Personne"** :

Homme - modification attribut classe mère
```
# modification de l'attribut sexe hérité
has '+sexe' => ( default => 'M');
```

En rajoutant un signe **+**, on peut rajouter une option à l'attribut *sexe* défini dans la classe **"Personne"**. Elle aura ainsi par défaut la valeur *'M'* (ou *'F'* pour la classe **"Femme"**).
Redéfinissons maintenant notre méthode **parler**. Juste pour rappel, la classe **"Homme"** redéfinit la méthode **parler** de la classe **"Personne"** dont elle hérite.

Homme - redéfinition méthode parler
```
# Méthode parler (un argument) qui va redéfinir la méthode parler
# de la classe mère
around 'parler' => sub {
  # Référence vers la méthode parler de la classe mère
  # objet et argument
  my ( $Ref_methode_parler_class, $this, $message ) = @_;

  unless ( defined $message ) {
    $message = 'je suis un homme et Ne sais pas quoi dire';
  }
  $this->$Ref_methode_parler_class( $message . ' (je suis un homme)' );

  return 1;
};
```

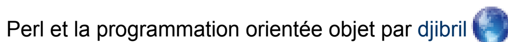
Nous utilisons ici le mot-clef **"around"**. Il permet d'envelopper l'appel de la méthode **parler**. Cette ligne ***around 'parler' => sub {};*** permet de redéfinir une méthode.

*Le premier argument correspond à la référence de la méthode originelle (**parler**). Les deux autres arguments sont l'objet (**$this**) et l'argument passé à la procédure.*

Pour la suite, pas de changement, on utilisera toujours la pseudoclasse **SUPER** si besoin, ainsi que la méthode **DEMOLISH** (pour le destructeur). Il n'est pas nécessaire d'appeler les méthodes **DEMOLISH** des classes mères, Moose le fait pour nous. Voici nos classes :

Homme
```
package Homme;     # Nom du package, de notre classe
use Carp;          # Utile pour émettre certains avertissements

use vars qw($VERSION);     # Version de notre module
$VERSION = '1.30';

use Moose;            # Il charge automatiquement strict et warnings
extends qw ( Personne Activite );    # hérite de la classe Personne et Activite

has barbu => (
  is       => 'rw',
  isa      => 'Int',
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.



Homme

```
    required => 0,
    reader   => 'obtenir_barbu',
    writer   => 'modifier_barbu',
    trigger  => sub {
      my ( $this, $barbu ) = @_;
      unless ( $barbu =~ /^0|1$/ ) {
        die("[Attention] Mauvais attribut barbu : $barbu [0 ou 1]\n");
      }
    },
);

# modification de l'attribut sexe hérité
has '+sexe' => ( default => 'M');

# Méthode parler (un argument) qui va redéfinir la méthode parler
# de la classe mère
around 'parler' => sub {
  # Référence vers la méthode parler de la classe mère
  # objet et argument
  my ( $Ref_methode_parler_class, $this, $message ) = @_;

  unless ( defined $message ) {
    $message = 'je suis un homme et Ne sais pas quoi dire';
  }
  $this->$Ref_methode_parler_class( $message . ' (je suis un homme)' );

  return 1;
};

# Destructeur
sub DEMOLISH {
  my $this = shift;

  # Avertissement que l'objet va mourir
  print "==================================================\n";
  print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
  print "==================================================\n";

  return;
}

1;    # Important, à ne pas oublier
__END__            # Le compilateur ne lira pas les lignes suivantes
```

Femme

```
package Femme;     # Nom du package, de notre classe
use Carp;          # Utile pour émettre certains avertissements

use vars qw($VERSION);    # Version de notre module
$VERSION = '1.30';

use Moose;             # Il charge automatiquement strict et warnings
extends qw ( Personne Activite );    # hérite de la classe Personne et Activite

# modification de l'attribut hérité
has '+sexe' => ( default => 'F');

# méthode accouche
sub accouche {
  my ( $this, $nombre_enfant ) = @_;

  if ( defined $nombre_enfant ) {
    unless ( $nombre_enfant =~ m{^\d+$} ) {
      croak "Mauvais argument : $nombre_enfant\n";
      return;
    }

    $this->parler("accouche de $nombre_enfant enfant(s)");
    my $NbrEnfant = $this->obtenir_nombre_enfant() + $nombre_enfant;
    $this->modifier_nombre_enfant($NbrEnfant);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

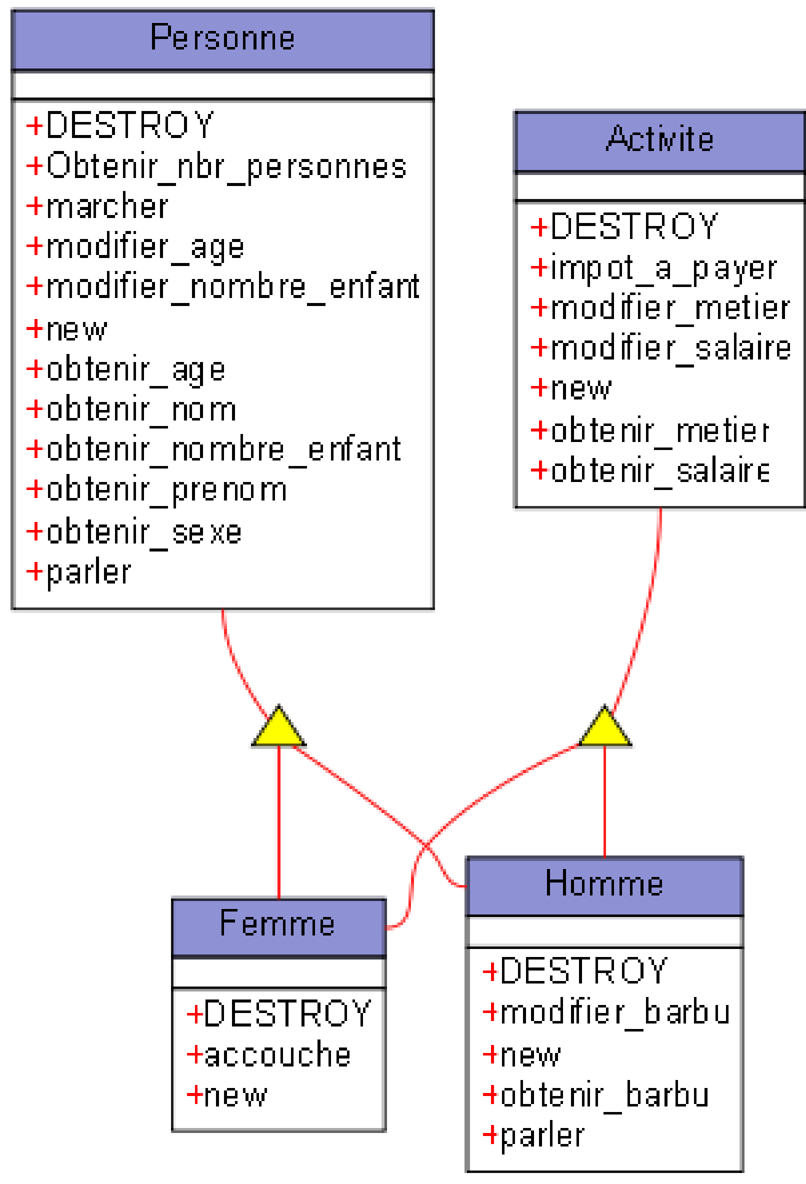
Femme
```
    }

    return;
}

# Destructeur
sub DEMOLISH {
    my $this = shift;

    # Avertissement que l'objet va mourir
    print "====================================================\n";
    print "L'objet de la classe " . __PACKAGE__ . " va mourir\n";
    print "====================================================\n";

    return;
}
1;    # Important, à ne pas oublier
__END__          # Le compilateur ne lira pas les lignes suivantes
```

Voilà, je ne vous ai exposé qu'une infime partie des fonctionnalités de Moose, car pour notre exemple, c'était suffisant. Sachez qu'avec Moose, il est possible de faire beaucoup plus. Vous pouvez jeter une attention particulière sur les modules :

- **Moose::Role** : Moose supporte les rôles qui sont des classes particulières, non instanciables ;
- les rôles sont une alternative plus puissante aux interfaces à la Java. Ils représentent une alternative plus sûre et plus flexible à l'héritage multiple ;
- **Moose::Cookbook** : vous y trouverez des exemples de code ;
- **Moose::Manual** : les manuels Moose.

Vous pouvez télécharger tous les scripts **Moose** de cet article **ici**.
Voici le modèle des classes

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.



Classes construites avec Moose

C-2 - Moose - Les inconvénients

- Le premier inconvénient de Moose est qu'il utilise beaucoup de modules du CPAN. De ce fait, son installation peut être difficile dans certains environnements.
- Il a un coût à la compilation, étant donné qu'il charge plusieurs modules.
- Les performances sont meilleures avec Perl 5.10 qui supporte mieux ces nouveaux modèles objets.
- Moose a pour but de vous faciliter l'écriture de Perl POO. Mais son but n'est pas de protéger l'encapsulation des données comme le font certains modules. Donc il est possible via **Data::Dumper** de visualiser le contenu de l'objet et de modifier son contenu (les valeurs des attributs) comme expliqué en début d'article.

Malgré ces inconvénients, il reste la référence en termes de POO avec Perl et il a un bel avenir devant lui.

C-3 - Coat - le petit dernier

Coat est un petit module qui a été créé pour pallier la lourdeur des dépendances de Moose. Il est purement Perl. Attention, ce n'est pas Moose, mais la syntaxe est identique. Ce module permet de faire de la POO comme Moose, mais il ne dispose pas de toutes les fonctionnalités de ce dernier. Ainsi, un code écrit en Coat peut être tout simplement

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

converti en Moose en changeant **use Coat** par **use Moose**. Il peut vous servir si vous n'arrivez pas à installer correctement Moose. Pour en savoir plus, le **CPAN** est votre ami.

D - Les techniques avancées, les objets inversés

A ce stade de l'article, vous maîtrisez les bases de la programmation orientée objet en Perl, les notions de classe, de méthode, de constructeur, la construction d'objets, etc. Vous êtes même capable de concevoir des classes de façon moderne avec Moose.
Vous avez remarqué que l'encapsulation est très limitée en Perl. Maintenant, nous verrons qu'il existe des modules nous permettant de mieux protéger l'implémentation de nos classes, c'est-à-dire de protéger l'accès à nos objets (qui ne sont que des hachages).

Comme vous le savez, Perl est certes permissif, mais tout est possible, à part faire le café ! Et vous connaissez sa devise, *"there is more than one way to do it"* (**TIMTOWTDI**).

D-1 - Notion d'objet inversé

Il existe une technique simple à mettre en place pour empêcher un script d'accéder au contenu des objets. Il s'agit de créer des objets inversés (ou *inside-out object*). Il s'agit d'un mode de conception qui est un peu différent de ce qu'on a pu voir jusqu'à présent dans cet article. Pour vous illustrer cette nouvelle notion, nous allons étudier trois modules du CPAN :

1. **Class::Std::Utils** aujourd'hui non recommandé ;
2. **Class::InsideOut** plus adapté et robuste que le précédent ;
3. **Object::InsideOut** encore plus puissant et rapide que Class::InsideOut.

Si vous rencontrez des difficultés pour installer ces modules, consultez cet **article**, et si ça ne vous aide pas, le forum Perl est là pour vous.

D-2 - Class::Std::Utils

Ce module est l'un des premiers à avoir été utilisé pour la création d'objets inversés. Aujourd'hui, il n'est plus recommandé, mais dans un but pédagogique, il est important de comprendre son fonctionnement.
Pour créer des objets inversés, voici quelques changements dans la création de nos objets à savoir :

1. tous nos attributs (nom et valeur) de notre classe ne seront plus stockés dans un seul hachage ;
2. le nom de chaque attribut ne sera donc plus la clé d'un hachage ;
3. le code entier d'une classe sera dans un bloc ;
4. chaque attribut a son hachage ;
5. dans le constructeur, l'objet inversé ne sera plus une référence anonyme à un hachage vide, mais à un scalaire anonyme ;
6. ...

Commençons par créer la classe **"Personne"**.

Personne.pm
```
package Personne;     # Nom du package, de notre classe
use warnings;         # Avertissement des messages d'erreurs
use strict;           # Vérification des déclarations
use Carp;             # Utile pour émettre certains avertissements

use vars qw($VERSION);    # Version de notre module
$VERSION = '1.10';

use Class::Std::Utils;    # Permet la création d'objets inversés
{
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Personne.pm
```
  # Tout le code de notre classe
}

1;    # Important, à ne pas oublier
__END__            # Le compilateur ne lira pas les lignes suivantes
```

Le bloc permet de limiter la portée des variables de notre classe. Créons maintenant un hachage par attribut.

Personne.pm
```
my ( %nom, %prenom, %sexe, %age, %nombre_enfant );
```

Créons notre constructeur.

Personne.pm - constructeur
```
# Constructeur de la classe Personne
  sub new {
    my ( $classe, $ref_arguments ) = @_;

    # Vérifions la classe
    $classe = ref($classe) || $classe;

    # Création d'un scalaire anonyme futur objet
    my $this = bless( anon_scalar(), $classe );

    # Vérification des champs nom, prénom et sexe
    foreach my $attribut (qw/ nom prenom sexe /) {
      unless ( defined $ref_arguments->{$attribut} ) {
        croak("[Attention] Attribut $attribut manquant\n");
      }
    }
    unless ( $ref_arguments->{sexe} =~ m{^M|F$}i ) {
      croak(
        "[Attention] Mauvais attribut sexe $ref_arguments->{sexe}, F ou M\n");
    }

    $nom{ ident $this}            = $ref_arguments->{nom};
    $prenom{ ident $this}         = $ref_arguments->{prenom};
    $sexe{ ident $this}           = $ref_arguments->{sexe};
    $age{ ident $this}            = $ref_arguments->{age};
    $nombre_enfant{ ident $this} = $ref_arguments->{nombre_enfant} || 0;

    # Nombre de personnes créées.
    $NbrPersonnes++;

    return $this;
  }
```

Au lieu de créer une référence anonyme à un hachage, nous avons créé un scalaire anonyme via la méthode **anon_scalar** exportée du module **Class::Std::Utils**. Elle a ensuite été bénie et liée à la classe.

Personne.pm - bénédiction et création d'objet
```
my $this = bless( anon_scalar(), $classe );
```

Une fois notre objet créé, il sert à créer une clé unique grâce à la méthode **ident** exportée du module **Class::Std::Utils** (on peut également utiliser la méthode **refaddr** du module Scalar::Util). En écrivant :

```
ident $this
```

Un entier unique est créé et est utilisé comme clé pour chacun des hachages de nos attributs qui sont inaccessibles de l'extérieur de la classe. Cet entier est unique et correspond à l'adresse mémoire où est stocké l'objet. Faisons un test :

Script principal
```
#!/usr/bin/perl
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Script principal
```
use warnings;
use strict;
use Carp;

use Personne;
my $Personne = Personne->new(
  { nom    => 'Henry',
    prenom => 'Antoine',
    sexe   => 'M',
  }
);

use Data::Dumper;
print Dumper $Personne;
```

Résultat
```
$VAR1 = bless( do{\(my $o = undef)}, 'Personne' );
```

Voilà, vous remarquez qu'il est impossible de voir le contenu des attributs, notre implémentation est donc mieux protégée. Notre objet ne contient que le scalaire anonyme. Voici un exemple de méthode **obtenir_nom**.

Personne.pm
```
# accesseur obtenir_nom
sub obtenir_nom {
  my $this = shift;
  return $nom{ ident $this};
}
```

En ce qui concerne les destructeurs, n'oubliez pas de détruire les clés de vos hachages.

Personne.pm - destructeur
```
# Destructeur
  sub DESTROY {
    my $this = shift;

    delete $nom{ ident $this};
    delete $prenom{ ident $this};
    delete $sexe{ ident $this};
    delete $age{ ident $this};
    delete $nombre_enfant{ ident $this};

    # Diminuer le nombre de personnes créées.
    $NbrPersonnes--;

    return;
  }
```

Dans les constructeurs des classes, ne bénissez pas l'objet reçu de la classe mère sous peine de ne plus pouvoir accéder aux attributs de celle-ci.
Voilà, j'espère que vous avez compris la notion d'objet inversé via ce module qui, je le rappelle, n'est plus conseillé.

Vous trouverez tous les scripts utilisant ce module **ICI**.

D-3 - Class::InsideOut

Ce module permet de protéger nos implémentations en utilisant toujours la notion d'objets inversés. Il nous facilite la vie dans la conception des accesseurs et mutateurs. Il est donc possible de rendre nos attributs publics, privés ou juste en lecture. Reprenons notre classe **"Personne"**.

Personne.pm
```
package Personne;    # Nom du package, de notre classe
use warnings;        # Avertissement des messages d'erreurs
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Personne.pm

```
use strict;          # Vérification des déclarations
use Carp;            # Utile pour émettre certains avertissements

use vars qw($VERSION);    # Version de notre module
$VERSION = '1.20';

use Class::InsideOut qw/ :std /;
{
  # Tout le code ici
}
1;
__END__
```

Nous appelons le module **Class::InsideOut** avec le tag **:std**. Cela nous permet d'exporter des fonctionnalités nous protégeront nos attributs comme nous le verrons par la suite. La création des attributs nécessite toujours un hachage par attribut. Mais l'écriture est différente et peu commune.

Personne.pm

```
use Class::InsideOut qw/ :std /;
{
  readonly nom    => my %nom;
  readonly prenom => my %prenom;
  readonly sexe   => my %sexe;
  public   age => my %age;
  public   nombre_enfant => my %nombre_enfant;

  # Comptage du nombre de personnes créées
  my $NbrPersonnes = 0;    # champ statique privé

  sub new {
    # ...
  }
```

Vous l'avez sûrement compris au premier regard, les attributs *nom*, *prenom* et *sexe* sont en lecture seule, alors que les attributs *age* et *nom_enfant* sont publics. Le module crée automatiquement des méthodes *nom*, *age*... qui seront des mutateurs/accesseurs. Pour lire le contenu, il suffit d'appeler la méthode *age* par exemple sans argument. Pour la modifier, il suffit de fournir un argument. Avant de continuer les explications, construisons notre constructeur.

Personne.pm - constructeur

```
sub new {
    my ( $classe, $ref_arguments ) = @_;

    # Vérifions la classe
    $classe = ref($classe) || $classe;

    # Création de notre objet
    my $this = register($classe);

    unless ( $ref_arguments->{sexe} =~ m{^M|F$}i ) {
      croak(
        "[Attention] Mauvais attribut sexe : $ref_arguments->{sexe}, F ou M\n");
    }

    $nom{ id $this}           = $ref_arguments->{nom};
    $prenom{ id $this}        = $ref_arguments->{prenom};
    $sexe{ id $this}          = $ref_arguments->{sexe};
    $age{ id $this}           = $ref_arguments->{age};
    $nombre_enfant{ id $this} = $ref_arguments->{nombre_enfant} || 0;

    # Nombre de personnes créées.
    $NbrPersonnes++;

    return $this;
  }
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

La fonction **register** du module permet de bénir le scalaire anonyme et de créer l'objet de façon sécurisée. De plus, il permet de rendre la classe **thread-safe** (*il est possible d'utiliser les threads, notamment faire du partage entre threads sans danger*).
On note l'utilisation de la fonction **id** identique à la façon **ident**. Elle n'est en fait qu'un alias de la fonction **refaddr** du module *Scalar::Util::refaddr*.
La conception des autres méthodes ne change pas. Il suffit de remplacer **ident** par **id**. Si nous souhaitons créer par exemple les accesseurs et mutateurs (obtenir_age et modifier_age), il suffit de mettre l'attribut *age* en **private** et de créer les deux méthodes.
En ce qui concerne les destructeurs, le module se charge de les appeler pour vous. Néanmoins, si vous souhaitez y faire appel pour x raisons (fermeture des connexions réseau, fichier...), il faut utiliser la méthode **DEMOLISH** mise à notre disposition par le module. Dans l'héritage multiple, vous ne devez pas l'appeler pour vos classes mères car Moose le fait pour nous.
Voici notre classe **"Personne"**.

```
package Personne;    # Nom du package, de notre classe
use warnings;        # Avertissement des messages d'erreurs
use strict;          # Vérification des déclarations
use Carp;            # Utile pour émettre certains avertissements

use vars qw($VERSION);     # Version de notre module
$VERSION = '1.20';

use Class::InsideOut qw/ :std /;
{
  private nom            => my %nom;
  private prenom         => my %prenom;
  private sexe           => my %sexe;
  private age            => my %age;
  private nombre_enfant => my %nombre_enfant;

  # Comptage du nombre de personnes créées
  my $NbrPersonnes = 0;     # champ statique privé

  sub new {
    my ( $classe, $ref_arguments ) = @_;

    # Vérifions la classe
    $classe = ref($classe) || $classe;

    # Création de notre objet
    my $this = register($classe);

    unless ( $ref_arguments->{sexe} =~ m{^M|F$}i ) {
      croak(
        "[Attention] Mauvais attribut sexe : $ref_arguments->{sexe}, F ou M\n");
    }

    $nom{ id $this}            = $ref_arguments->{nom};
    $prenom{ id $this}         = $ref_arguments->{prenom};
    $sexe{ id $this}           = $ref_arguments->{sexe};
    $age{ id $this}            = $ref_arguments->{age};
    $nombre_enfant{ id $this} = $ref_arguments->{nombre_enfant} || 0;

    # Nombre de personnes créées.
    $NbrPersonnes++;

    return $this;
  }

  # accesseur obtenir_nom
  sub obtenir_nom {
    my $this = shift;
    return $nom{ id $this};
  }

  # accesseur obtenir_prenom
  sub obtenir_prenom {
    my $this = shift;
    return $prenom{ id $this};
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

```
}

# accesseur obtenir_sexe
sub obtenir_sexe {
  my $this = shift;
  $sexe{ id $this};
}

# accesseur obtenir_age
sub obtenir_age {
  my $this = shift;
  return $age{ id $this};
}

# mutateur modifier_age
sub modifier_age {
  my ( $this, $age ) = @_;

  if ( defined $age ) {
    unless ( $age =~ m{^\d+$} ) {
      croak "Mauvais argument : $age\n";
      return;
    }
    $age{ id $this} = $age;
  }

  return;
}

# accesseur obtenir_nombre_enfant
sub obtenir_nombre_enfant {
  my $this = shift;

  return $nombre_enfant{ id $this};
}

# mutateur modifier_nombre_enfant
sub modifier_nombre_enfant {
  my ( $this, $nombre_enfant ) = @_;

  if ( defined $nombre_enfant ) {
    unless ( $nombre_enfant =~ m{^\d+$} ) {
      croak "Mauvais argument : $nombre_enfant\n";
      return;
    }
    $nombre_enfant{ id $this} = $nombre_enfant;
  }

  return;
}

# Méthode marcher - ne prend aucun argument
sub marcher {
  my $this = shift;

  my ( $nom, $prenom ) = ( $nom{ id $this}, $prenom{ id $this} );
  print "[$nom $prenom] Marche\n";

  return;
}

# Méthode parler - un argument
sub parler {
  my ( $this, $message ) = @_;

  unless ( defined $message ) {
    $message = 'Ne sais pas quoi dire';
  }

  # Le premier caractère de la phrase est mis en majuscule
  $message = ucfirst($message);
  my ( $nom, $prenom ) = ( $nom{ id $this}, $prenom{ id $this} );
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

```
    print "[$nom $prenom] $message\n";

    return 1;
  }

  # méthode de classe Obtenir_nbr_personnes
  sub Obtenir_nbr_personnes {
    my $classe = shift;

    return $NbrPersonnes;
  }

  # Destructeur bis
  sub DEMOLISH {
    my $this = shift;
    #  print "Fin classe : ", __PACKAGE__,"\n";
  }

}

1;    # Important, à ne pas oublier
__END__           # Le compilateur ne lira pas les lignes suivantes
```

Vous trouverez tous les scripts utilisant ce module **ici**.

D-4 - Object::InsideOut

Ce module est le plus puissant des trois. Nous allons voir comment l'utiliser à travers notre classe **"Personne"**. Commençons par la créer.

Classe Personne

```
package Personne;
{    # Nom du package, de notre classe

  use warnings;    # Avertissement des messages d'erreurs
  use strict;      # Vérification des déclarations
  use Carp;        # Utile pour émettre certains avertissements

  use vars qw($VERSION);    # Version de notre module
  $VERSION = '2.0';

  use Object::InsideOut;    # Module pour la création d'objets inversés

  # Tout notre code

}

1;                          # Important, à ne pas oublier
__END__           # Le compilateur ne lira pas les lignes suivantes
```

Maintenant, passons aux attributs. Déclarons les cinq attributs habituels avec en prime une batterie de tests sur les valeurs des attributs qui sont passées au constructeur *new* de notre classe **"Personne"**. Notez que l'on n'a pas besoin de créer un constructeur, **Object::InsideOut** le fait pour nous.

Classe Personne - Attributs

```
# Déclaration des attributs
  # Nom - attribut avec uniquement accesseur
  my @nom :Field
          :Get(obtenir_nom)                         # Accesseur
          :Arg('Name' => 'nom', 'Mandatory' => 1)    # Attribut obligatoire
          :Type('scalar')                            # Type argument
          ;

  # Prenom - attribut avec uniquement accesseur
  my @prenom :Field
             :Get(obtenir_prenom)
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe Personne - Attributs

```
              :Arg('Name' => 'prenom', 'Mandatory' => 1)
              :Type('scalar')
              ;

  # Sexe - attribut avec uniquement accesseur
  my @sexe :Field
            :Get(obtenir_sexe)
            :Arg('Name' => 'sexe', 'Mandatory' => 1)
            :Type('scalar')
            ;

  # nombre_enfant - avec accesseur et mutateur et valeur par défaut
  # on n'utilise pas Arg car on va faire une vérification
  my @nombre_enfant :Field
                     :Get(obtenir_nombre_enfant)
                     :Set(modifier_nombre_enfant)  # Mutateur
                     :Default(0)                   # Valeur par défaut
                     :Type(numeric);

  # Age - attribut avec accesseur et mutateur
  # on n'utilise pas Arg car on va faire une vérification
  my @age :Field
           :Get(obtenir_age)
           :Set(modifier_age) # Mutateur
           :Type(numeric)
           ;
  # Vérification sur l'age et le nombre d'enfants
  my %NosAttributs : InitArgs = (
    'nombre_enfant' => '',
    'age' => '',
  );

  sub _init : Init {
    my ( $self, $args ) = @_;

    foreach my $NomAttribut ( qw/ age nombre_enfant / ) {
      if ( exists $args->{$NomAttribut} ) {
        if ( ( $NomAttribut eq 'nombre_enfant') and $args->{$NomAttribut} >= 0 ) {
          # Déclaration de l'attribut
          $self->set( \@nombre_enfant, $args->{$NomAttribut} );
        }
        elsif ( ( $NomAttribut eq 'age') and $args->{$NomAttribut} > 0 ) {
          # Déclaration de l'attribut
          $self->set( \@age, $args->{$NomAttribut} )  if ( $NomAttribut eq 'age');
        }
        else {
          die("Attribut $NomAttribut ($args->{$NomAttribut}) doit être >= 0\n") if
 ( $NomAttribut eq 'nombre_enfant');
          die("Attribut $NomAttribut ($args->{$NomAttribut}) doit être > 0\n");
        }
      }
    }
  }
```

Vous devez vous dire que l'écriture est un peu bizarre, je sais. N'ayez pas peur, au début c'est effrayant, une fois qu'on a compris comment ça fonctionne, c'est formidable !
Le premier avantage de ce module est que l'on va plutôt utiliser des tableaux à la place des hachages habituels pour chaque attribut. On y gagne énormément en performance (plus de 40% d'après l'auteur). Il est néanmoins toujours possible d'utiliser des hachages, mais ce n'est pas conseillé.
Les variables sont ensuite taguées. Il existe beaucoup de tags dans la documentation. Voici une explication des tags utilisés ci-dessus :
:Field permet de déclarer notre attribut. Il crée automatiquement un index au moyen de l'entier unique qui sera créé via l'objet (c'est ce que l'on a vu dans les autres modules ci-dessus). Il fait tout pour nous et on n'a plus besoin de s'en soucier ;
:Get et **:Set** permettent de créer des accesseurs et mutateurs de noms différents, sinon, **Object::InsideOut** crée automatiquement un accesseur/mutateur du même nom que celui de l'attribut ;

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

:Type permet de vérifier le type d'argument auquel le constructeur doit s'attendre. Par exemple pour notre attribut *age*, le fait d'écrire **:Type(numeric)** permet de vérifier que l'âge donné sera bien un nombre ;
:Default permet d'attribuer une valeur par défaut si l'utilisateur ne spécifie rien du tout ;
:Arg permet de donner un nom à l'attribut que l'on attend de l'utilisateur. Ce tag est obligatoire. S'il n'est pas donné, il faudra utiliser une autre méthode que l'on aborde plus bas dans l'article (ex: attributs *nombre_enfant* et *age*). Il est possible de l'écrire de deux façons différentes :

Tag :Arg
```
:Arg('Name' => 'NomAttribut')
# ou
:Arg(NomAttribut)
```

Les deux notations sont identiques. La première est utile si l'on souhaite utiliser d'autres options internes à :Arg. C'est le cas de notre attribut *prenom*.

```
:Arg('Name' => 'prenom', 'Mandatory' => 1)
```

L'option *Mandatory* est utilisée dans le but d'obliger l'utilisateur à donner en argument son prénom.
Si vous ne souhaitez pas utiliser le tag **:Arg**, il est possible de déclarer l'attribut différemment. C'est le cas pour nos attributs *age* et *nombre_enfant*. Nous souhaitons vérifier que l'utilisateur donne un nombre (c'est déjà fait via le tag Type(numeric)), qui sera plus grand ou égal à 0.
Créons un hachage dans lequel les clés seront les noms des attributs à créer.

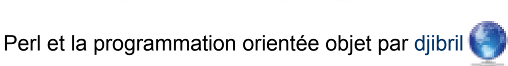
```
my %NosAttributs : InitArgs = (
    'nombre_enfant' => '',
    'age' => '',
  );
```

Remarquez que ce hachage doit être tagué de **: InitArgs**. Ce tag permet au module d'extraire dans les arguments fournis à new ceux qui correspondent à l'âge et au nombre d'enfants. Une fois ce hachage créé, on place une subroutine (une fonction) qui sera taguée de **: Init**. Cela permet au module d'exécuter une tâche quand les attributs sont initialisés. Dans notre cas, on vérifie à ce moment que l'âge ou le nombre d'enfants est renseigné. Si c'est le cas, on vérifie qu'ils sont plus grands (ou égal - pour le nombre d'enfants) que 0 et on modifie son contenu qui était initialisé à rien **''** via la méthode **set();** :

```
sub _init : Init {
    my ( $self, $args ) = @_;

    foreach my $NomAttribut ( qw/ age nombre_enfant / ) {
      if ( exists $args->{$NomAttribut} ) {
        if ( ( $NomAttribut eq 'nombre_enfant') and $args->{$NomAttribut} >= 0 ) {
          # Déclaration de l'attribut
          $self->set( \@nombre_enfant, $args->{$NomAttribut} );
        }
        elsif ( ( $NomAttribut eq 'age') and $args->{$NomAttribut} > 0 ) {
          # Déclaration de l'attribut
          $self->set( \@age, $args->{$NomAttribut} )  if ( $NomAttribut eq 'age');
        }
        else {
          die("Attribut $NomAttribut ($args->{$NomAttribut}) doit être >= 0\n") if
 ( $NomAttribut eq 'nombre_enfant');
          die("Attribut $NomAttribut ($args->{$NomAttribut}) doit être > 0\n");
        }
      }
    }
  }
```

En ce qui concerne les méthodes, elles sont écrites comme on en a l'habitude. Mais le module nous permet de les rendre publiques (par défaut), privées ou restreintes à certaines classes pendant l'héritage. Lisez la documentation, elle est vraiment riche. Il est également possible de créer une méthode taguée d'un **:PreInit** qui sera cachée, et exécutée avant l'initialisation des variables. Créer une telle méthode est généralement utile pour définir des attributs

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.



dans les classes mères dont on hérite. Nous allons l'utiliser pour incrémenter notre variable comptant le nombre de personnes créées.

méthodes
```
# Incrémentation du nombre de personnes créées.
  sub _AjouterPersonnes :PreInit {
        my ($self, $args) = @_;
        $NbrPersonnes ++;
  }

  # Méthode marcher - ne prend aucun argument
  sub marcher {
    my $this = shift;
    my ( $nom, $prenom ) = ( $this->obtenir_nom, $this->obtenir_prenom );
    print "[$nom $prenom] Marche\n";

    return;
  }

  # Méthode parler - un argument
  sub parler {
    my ( $this, $message ) = @_;

    unless ( defined $message ) {
      $message = 'Ne sais pas quoi dire';
    }

    # Le premier caractère de la phrase est mis en majuscule
    $message = ucfirst($message);
    my ( $nom, $prenom ) = ( $this->obtenir_nom, $this->obtenir_prenom );
    print "[$nom $prenom] $message\n";

    return 1;
  }

  # méthode de classe Obtenir_nbr_personnes
  sub Obtenir_nbr_personnes {
    my $classe = shift;

    return $NbrPersonnes;
  }
```

Pour le destructeur, **Object::InsideOut** le crée pour nous. Mais comme nous pouvons avoir besoin d'effectuer des tâches avant la destruction de l'objet, il nous donne la possibilité d'exécuter une méthode destructrice en amont de DESTROY. Il suffit de créer une méthode taguée avec **:Destroy**. Le nom de la méthode est au choix de l'utilisateur.

Personne - destructeur
```
# Destructeur
  sub _destroy :Destroy {
    my $this = shift;
    print "Fin classe : ", __PACKAGE__,"\n";

    # Diminuer le nombre de personnes créées.
    $NbrPersonnes--;

  }
```

Affichons notre classe **"Personne"** pour avoir une vue d'ensemble.

Classe Personne
```
package Personne;
{    # Nom du package, de notre classe

  use warnings;    # Avertissement des messages d'erreurs
  use strict;      # Vérification des déclarations
  use Carp;        # Utile pour émettre certains avertissements

  use vars qw($VERSION);    # Version de notre module
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe Personne

```
$VERSION = '2.0';

use Object::InsideOut;    # Module pour la création d'objets inversés

# Comptage du nombre de personnes créées
my $NbrPersonnes = 0;     # champ statique privé

# Déclaration des attributs
# Nom - attribut avec uniquement accesseur
my @nom :Field
        :Get(obtenir_nom)                         # Accesseur
        :Arg('Name' => 'nom', 'Mandatory' => 1)   # Attribut obligatoire
        :Type('scalar')                           # Type argument
        ;

# Prenom - attribut avec uniquement accesseur
my @prenom :Field
           :Get(obtenir_prenom)
           :Arg('Name' => 'prenom', 'Mandatory' => 1)
           :Type('scalar')
           ;

# Sexe - attribut avec uniquement accesseur
my @sexe :Field
         :Get(obtenir_sexe)
         :Arg('Name' => 'sexe', 'Mandatory' => 1)
         :Type('scalar')
         ;

# nombre_enfant - avec accesseur et mutateur et valeur par défaut
# on n'utilise pas Arg car on va faire une vérification
my @nombre_enfant :Field
                  :Get(obtenir_nombre_enfant)
                  :Set(modifier_nombre_enfant)  # Mutateur
                  :Default(0)                   # Valeur par défaut
                  :Type(numeric);

# Age - attribut avec accesseur et mutateur
# on n'utilise pas Arg car on va faire une vérification
my @age :Field
        :Get(obtenir_age)
        :Set(modifier_age) # Mutateur
        :Type(numeric)
        ;
# Vérification sur l'age et le nombre d'enfants
my %NosAttributs : InitArgs = (
  'nombre_enfant' => '',
  'age' => '',
);

sub _init : Init {
  my ( $self, $args ) = @_;

  foreach my $NomAttribut ( qw/ age nombre_enfant / ) {
    if ( exists $args->{$NomAttribut} ) {
      if ( ( $NomAttribut eq 'nombre_enfant') and $args->{$NomAttribut} >= 0 ) {
        # Déclaration de l'attribut
        $self->set( \@nombre_enfant, $args->{$NomAttribut} );
      }
      elsif ( ( $NomAttribut eq 'age') and $args->{$NomAttribut} > 0 ) {
        # Déclaration de l'attribut
        $self->set( \@age, $args->{$NomAttribut} )  if ( $NomAttribut eq 'age');
      }
      else {
        die("Attribut $NomAttribut ($args->{$NomAttribut}) doit être >= 0\n") if
( $NomAttribut eq 'nombre_enfant');
        die("Attribut $NomAttribut ($args->{$NomAttribut}) doit être > 0\n");
      }
    }
  }
}
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe Personne

```
    # Incrémentation du nombre de personnes créées.
    sub _AjouterPersonnes :PreInit {
           my ($self, $args) = @_;
           $NbrPersonnes ++;
    }


    # Méthode marcher - ne prend aucun argument
    sub marcher {
      my $this = shift;
      my ( $nom, $prenom ) = ( $this->obtenir_nom, $this->obtenir_prenom );
      print "[$nom $prenom] Marche\n";

      return;
    }

    # Méthode parler - un argument
    sub parler {
      my ( $this, $message ) = @_;

      unless ( defined $message ) {
        $message = 'Ne sais pas quoi dire';
      }

      # Le premier caractère de la phrase est mis en majuscule
      $message = ucfirst($message);
      my ( $nom, $prenom ) = ( $this->obtenir_nom, $this->obtenir_prenom );
      print "[$nom $prenom] $message\n";

      return 1;
    }

    # méthode de classe Obtenir_nbr_personnes
    sub Obtenir_nbr_personnes {
      my $classe = shift;

      return $NbrPersonnes;
    }

    # Destructeur
    sub _destroy :Destroy {
      my $this = shift;
      print "Fin classe : ", __PACKAGE__,"\n";

      # Diminuer le nombre de personnes créées.
      $NbrPersonnes--;

    }


}

1;    # Important, à ne pas oublier
__END__           # Le compilateur ne lira pas les lignes suivantes
```

Voilà, ce n'est pas compliqué j'espère !! Je ne vous ai pas parlé des classes **"Activite"**, **"Homme"**, **"Femme"** car le procédé est le même. Pour hériter des autres classes, il faut la ligne suivante :

```
use Object::InsideOut qw ( Personne Activite );
```

Et vous ne devez plus utiliser le module **base**. Object::InsideOut le fait à notre place. Voici la classe "Femme".

Classe Femme

```
package Femme;     # Nom du package, de notre classe
{
  use warnings;      # Avertissement des messages d'erreurs
  use strict;        # Vérification des déclarations
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Classe Femme

```
  use Carp;         # Utile pour émettre certains avertissements

  use vars qw($VERSION);    # Version de notre module
  $VERSION = '2.0';

  use Object::InsideOut qw ( Personne Activite );   # hérite de la classe Personne et Activite

  # inialisation du sexe féminin chez la personne
  sub pre_init :PreInit {
       my ($self, $args) = @_;
       my $sexe = 'F';
       if ( exists $args->{sexe} and $args->{sexe} !~ m{^$sexe$}i ) {
         carp("[Attention] : mauvais sexe $args->{sexe}\n");
       }

       $args->{sexe} = $sexe;
  }

  # méthode accouche
  sub accouche {
    my ( $this, $nombre_enfant ) = @_;

    if ( defined $nombre_enfant ) {
      unless ( $nombre_enfant =~ m{^\d+$} ) {
        croak "Mauvais argument : $nombre_enfant\n";
        return;
      }

      $this->parler("accouche de $nombre_enfant enfant(s)");
      my $NbrEnfant = $this->nombre_enfant() + $nombre_enfant;
      $this->nombre_enfant($NbrEnfant);
    }

    return;
  }

  # Destructeur
  sub _destroy :Destroy {
    my $this = shift;
    print "Fin classe : ", __PACKAGE__,"\n";
  }

}
1;                                          # Important, à ne pas oublier
__END__            # Le compilateur ne lira pas les lignes suivantes
```

Pour en savoir plus sur ce module, je vous recommande de lire sa documentation (**Object::InsideOut**)

Vous pouvez télécharger tous les scripts utilisés pour ce module **ICI**.

D-5 - Comparaison entre ces trois modules

Cet article vous a exposé trois modules permettant de créer des objets inversés afin de protéger votre implémentation. Vous avez constaté que c'est simple et assez facile à mettre place. Voici une petite comparaison des trois modules pour résumer :

- **Class::Std::Utils** est toujours fonctionnel mais non recommandé car à ce jour, d'autres modules font la même chose que lui plus proprement et simplement ;
- **Class::InsideOut** et **Object::InsideOut** sont les plus aboutis et recommandés ;
- **Class::InsideOut** est thread safe avec une version de Perl supérieure à 5.8.5 et permet la sérialisation des objets ;
- **Object::InsideOut** est thread safe aussi. Il est beaucoup plus complet dans la protection de l'implémentation. Il permet la sérialisation des objets avec storable. Il est plus performant et les fonctionnalités sont nombreuses.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Pour les utilisateurs de **Perl 5.10**, vous pouvez jeter un oeil sur module très intéressant pour la gestion des objets inversés : **Hash::Util::FieldHash**.

E - Conclusion

J'espère que cet article vous a aidé à comprendre Perl et la programmation orientée objet. Nous sommes partis des bases aux techniques les plus évoluées en passant par des modules innovant comme Moose. Vous avez pu constater qu'il est tout de même possible d'encapsuler correctement ses données. Si vous avez des remarques, questions, corrections ou souhaitez discuter, n'hésitez pas en vous rendant ici :

Vous pouvez télécharger tous les scripts de cet article **ICI**, sans oublier toutes les **images** des classes.

F - Références utilisées

- **Moose::Role**
- **Moose::Cookbook** : vous y trouverez des exemples de codes
- **Moose::Manual** : les manuels Moose
- **coat**
- **Class::ISA**
- **Class::Std::Utils**
- **Class::InsideOut**
- **Object::InsideOut**
- **Hash::Util::FieldHash**
- **Perl Best Practices** de Damian Conway
- **De l'art de programmer en Perl** traduction de *"Perl Best Practices"* de Damian Conway
- **Data::Dumper**
- **Linux Dossiers 2** (avril/mai/juin 2004), Sylvain Lhuillier
- **UML::Class::Simple** m'a permis de créer les modèles des classes

G - Remerciements

Je tiens à remercier les personnes suivantes pour la relecture de l'article, les rectifications, les corrections pertinentes et les rajouts :

- **stoyak**, l'oeil de lynx
- **jedai**, le pointilleux
- **Philou67430**, deuxième oeil de lynx
- **gorgonite**, le sémanticien
- **ClaudeLELOUP**, le rigoureux

Sans oublier l'équipe de developpez.com.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2011 djibril. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.