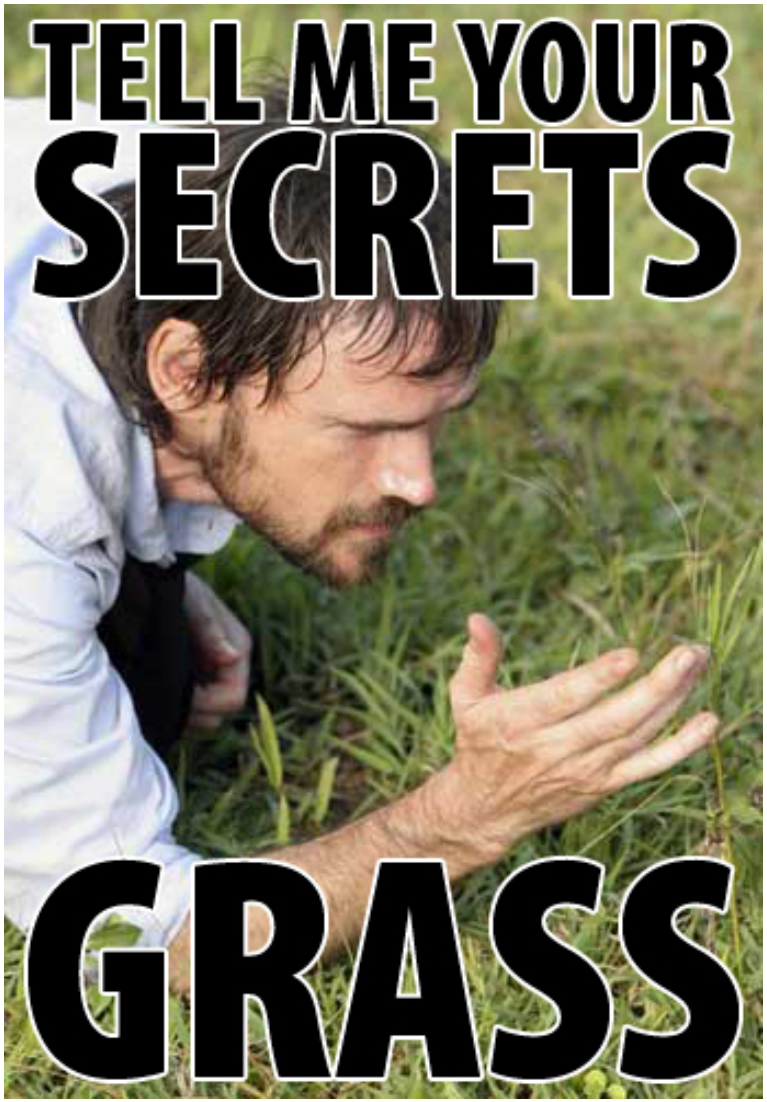
Osez dire
NON
à l'appel indirect de méthodes

Vincent Pit

2011-10-19

Différents appels de méthodes en Perl



Les objets selon le modèle standard de Perl

- les classes sont des espaces de noms ;
- les objets sont des références ;
- le lien référence-espace de nom est établi par `bless` ;
- les méthodes sont des fonctions.

Deux manières d'appeler une méthode

```
package X;

use strict;
use warnings;

sub new { bless { }, shift }
```

Deux manières d'appeler une méthode

```
package X;

use strict;
use warnings;

sub new { bless { }, shift }
```

- appel de méthode **direct** :

  ```
  my $obj = X->new;
  ```

Deux manières d'appeler une méthode

```
package X;

use strict;
use warnings;

sub new { bless { }, shift }
```

- appel de méthode **direct** :

  ```
  my $obj = X->new;
  ```

- appel de méthode **indirect** (ou datif) :

  ```
  my $obj = new X;
  ```

Deux manières d'appeler une méthode

```
package X;

use strict;
use warnings;

sub new { bless { }, shift }
```

- appel de méthode **direct** :

  `my $obj = X->new;` BIEN

- appel de méthode **indirect** (ou datif) :

  `my $obj = new X;` PABIEN

Deux manières d'appeler une fonction

- avec parenthèses :

```
my $x = zap();
my $y = zap(1, 2, 3);
```

Deux manières d'appeler une fonction

- avec parenthèses :

```
my $x = zap();
my $y = zap(1, 2, 3);
```

- sans parenthèses :

```
sub zap { } # définie avant
my $x = zap;
my $y = zap 1, 2, 3;
```

Les barewords

- Un *bareword* est un mot (`/^(?:[a-zA-Z_-]|::)/`) qui apparaît dans le code source hors d'une chaîne de caractères.

- Problème : comment Perl décide-t-il si

  ```
  my $obj = new X;
  ```

  signifie un appel de méthode ou de fonction ?

C'est l'heure du quiz

Question 1

```
$ cat q1.pl
#!perl
use 5.010; use strict; use warnings;

sub X::new { say "X::new: @_" }

my $obj = new X;

$ perl q1.pl
```

Question 1

```
$ cat q1.pl
#!perl
use 5.010; use strict; use warnings;

sub X::new { say "X::new: @_" }

my $obj = new X;

$ perl q1.pl
X::new: X
```

Question 2

```
$ cat q2.pl
#!perl
use 5.010; use strict; use warnings;

sub X::new { say "X::new: @_" }

sub new { say "main::new: @_" }

my $obj = new X;

$ perl q2.pl
```

Question 2

```
$ cat q2.pl
#!perl
use 5.010; use strict; use warnings;

sub X::new { say "X::new: @_" }

sub new { say "main::new: @_" }

my $obj = new X;

$ perl q2.pl
X::new: X
```

Question 2

```
$ cat q2.pl
#!perl
use 5.010; use strict; use warnings;

sub X::new { say "X::new: @_" }

sub new { say "main::new: @_" }

my $obj = X->new;

$ perl q2.pl
```

Question 2

```
$ cat q2.pl
#!perl
use 5.010; use strict; use warnings;

sub X::new { say "X::new: @_" }

sub new { say "main::new: @_" }

my $obj = X->new;

$ perl q2.pl
X::new: X
```

Question 2

```
$ cat q2.pl
#!perl
use 5.010; use strict; use warnings;

sub X::new { say "X::new: @_" }

sub new { say "main::new: @_" }

my $obj = new Y;

$ perl q2.pl
```

Question 2

```
$ cat q2.pl
#!perl
use 5.010; use strict; use warnings;

sub X::new { say "X::new: @_" }

sub new { say "main::new: @_" }

my $obj = new Y;

$ perl q2.pl
Bareword "Y" not allowed while "strict subs"
in use at q2.pl line 8.
Execution of q2.pl aborted due to compilation errors.
```

Question 2

```
$ cat q2.pl
#!perl
use 5.010; use strict; use warnings;

sub X::new { say "X::new: @_" }

sub new { say "main::new: @_" }

my $obj = Y->new;

$ perl q2.pl
```

Question 2

```
$ cat q2.pl
#!perl
use 5.010; use strict; use warnings;

sub X::new { say "X::new: @_" }

sub new { say "main::new: @_" }

my $obj = Y->new;

$ perl q2.pl
Can't locate object method "new" via package "Y"
(perhaps you forgot to load "Y"?) at q2.pl line 8.
```

Question 3

```
$ cat q3.pl
#!perl
use 5.010; use strict; use warnings;
{
 package X;
 sub new   { say "X::new: @_" }
 sub class { 'Y' }
}
sub Y::new { say "Y::new: @_" }

my $obj = new X->class ;

$ perl q3.pl
```

Question 3

```
$ cat q3.pl
#!perl
use 5.010; use strict; use warnings;
{
 package X;
 sub new   { say "X::new: @_" }
 sub class { 'Y' }
}
sub Y::new { say "Y::new: @_" }

my $obj = new X->class ;

$ perl q3.pl
X::new: X
Can't call method "class" without a package or object
reference at q3.pl line 10.
```

Question 3

```
$ cat q3.pl
#!perl
use 5.010; use strict; use warnings;
{
 package X;
 sub new   { say "X::new: @_" }
 sub class { 'Y' }
}
sub Y::new { say "Y::new: @_" }

my $obj = new X->class # = X->new->class;

$ perl q3.pl
```

Question 3

```
$ cat q3.pl
#!perl
use 5.010; use strict; use warnings;
{
 package X;
 sub new   { say "X::new: @_" }
 sub class { 'Y' }
}
sub Y::new { say "Y::new: @_" }

my $obj = X->class->new;

$ perl q3.pl
Y::new: Y
```

Question 4

```
$ cat q4.pl
#!perl
use 5.010; use strict; use warnings;
{
 package X;
 sub new { say "X::new: @_" }
 sub wat { my $class = shift; new $class }
}
```

Question 4

```
$ cat q4.pl
#!perl
use 5.010; use strict; use warnings;
{
 package X;
 sub new { say "X::new: @_" }
 sub wat { my $class = shift; new $class }
}
{
 package Y;
 our @ISA = 'X';
 sub new { say "Y::new: @_" }
}
my $obj = Y->wat;

$ perl q4.pl
```

Question 4

```
$ cat q4.pl
#!perl
use 5.010; use strict; use warnings;
{
 package X;
 sub new { say "X::new: @_" }
 sub wat { my $class = shift; new $class }
}
{
 package Y;
 our @ISA = 'X';
 sub new { say "Y::new: @_" }
}
my $obj = Y->wat;

$ perl q4.pl
X::new: Y
```

Question 4

```
$ cat q4.pl
#!perl
use 5.010; use strict; use warnings;
{
 package X;
 sub new { say "X::new: @_" }
 sub wat { my $class = shift; $class->new }
}
{
 package Y;
 our @ISA = 'X';
 sub new { say "Y::new: @_" }
}
my $obj = Y->wat;

$ perl q4.pl
```

Question 4

```
$ cat q4.pl
#!perl
use 5.010; use strict; use warnings;
{
 package X;
 sub new { say "X::new: @_" }
 sub wat { my $class = shift; $class->new }
}
{
 package Y;
 our @ISA = 'X';
 sub new { say "Y::new: @_" }
}
my $obj = Y->wat;

$ perl q4.pl
Y::new: Y
```

Question 5

```
$ cat q5.pl
#!perl
use 5.010; use strict; use warnings;
{
 package Tag;
 sub new {
  my $class = shift;
  bless { id => $_[0] }, $class
 }
 sub id { $_[0]->{id} }
}
```

Question 5

```
$ cat q5.pl
#!perl
use 5.010; use strict; use warnings;
{
 package Tag;
 sub new {
  my $class = shift;
  bless { id => $_[0] }, $class
 }
 sub id { $_[0]->{id} }
}
{
 package X;
 sub new {
  my $class = shift;
  bless { tag => $_[0] }, $class
 }
 sub id {
  my $self = shift;
  return id $self->{tag}
 }
}
```

Question 5

```
$ cat q5.pl
#!perl
use 5.010; use strict; use warnings;
{
 package Tag;
 sub new {
  my $class = shift;
  bless { id => $_[0] }, $class
 }
 sub id { $_[0]->{id} }
}

 my $tag = Tag->new(123);
 my $x = X->new($tag);
 say $x->id;

 $ perl q5.pl
```

```
{
 package X;
 sub new {
  my $class = shift;
  bless { tag => $_[0] }, $class
 }
 sub id {
  my $self = shift;
  return id $self->{tag}
 }
}
```

Question 5

```
$ cat q5.pl
#!perl
use 5.010; use strict; use warnings;
{
 package Tag;
 sub new {
  my $class = shift;
  bless { id => $_[0] }, $class
 }
 sub id { $_[0]->{id} }
}

 my $tag = Tag->new(123);
 my $x = X->new($tag);
 say $x->id;
```

```
{
 package X;
 sub new {
  my $class = shift;
  bless { tag => $_[0] }, $class
 }
 sub id {
  my $self = shift;
  return id $self->{tag}
 }
}
```

```
$ perl q5.pl
Deep recursion on subroutine "X::id" at q5.pl line 20.
```

Question 5

```
$ cat q5.pl
#!perl
use 5.010; use strict; use warnings;
{
 package Tag;
 sub new {
  my $class = shift;
  bless { id => $_[0] }, $class
 }
 sub id { $_[0]->{id} }
}

 my $tag = Tag->new(123);
 my $x = X->new($tag);
 say $x->id;

 $ perl q5.pl
```

```
{
 package X;
 sub new {
  my $class = shift;
  bless { tag => $_[0] }, $class
 }
 sub id {
  my $self = shift;
  return $self->{tag}->id
 }
}
```

Question 5

```
$ cat q5.pl
#!perl
use 5.010; use strict; use warnings;
{
 package Tag;
 sub new {
  my $class = shift;
  bless { id => $_[0] }, $class
 }
 sub id { $_[0]->{id} }
}

 my $tag = Tag->new(123);
 my $x = X->new($tag);
 say $x->id;

 $ perl q5.pl
 123
```

```
{
 package X;
 sub new {
  my $class = shift;
  bless { tag => $_[0] }, $class
 }
 sub id {
  my $self = shift;
  return $self->{tag}->id
 }
}
```

Question 6

```
$ cat q6.pl
use 5.010; use strict; use warnings;

my $result = parse { version => 1 };

$ perl q6.pl
```

Question 6

```
$ cat q6.pl
use 5.010; use strict; use warnings;

my $result = parse { version => 1 };

$ perl q6.pl
(rien !!!)
```

Question 6

```
$ cat q6.pl
use 5.010; use strict; use warnings;
use Data::Dumper;

my $result = parse { version => 1 };
print Dumper $result;

$ perl q6.pl
```

Question 6

```
$ cat q6.pl
use 5.010; use strict; use warnings;
use Data::Dumper;

my $result = parse { version => 1 };
print Dumper $result;

$ perl q6.pl
$VAR1 = bless( {
                 'original' => '1',
                 'version' => [
                                1
                               ]
               }, 'version' );
```

Question 6 bis

```
$ cat q6b.pl
use 5.010; use strict; use warnings;

sub alligator { say "main::alligator: @_" }

al1igator { miam => 1 };

$ perl q6b.pl
```

Question 6 bis

```
$ cat q6b.pl
use 5.010; use strict; use warnings;

sub alligator { say "main::alligator: @_" }

al1igator { miam => 1 };

$ perl q6b.pl
Can't locate object method "al1igator" via package
"miam" (perhaps you forgot to load "miam"?)
at q6b.pl line 6.
```

Question 7

```
$ cat One.pm
package One;
use strict; use warnings;
use Two;
sub one {
 Two::two {
  (foo => 'bar', baz => 'quux')
 }
}
1;
```

Question 7

```
$ cat One.pm
package One;
use strict; use warnings;
use Two;
sub one {
 Two::two {
  (foo => 'bar', baz => 'quux')
 }
}
1;
```

```
$ cat Two.pm
package Two;
use strict; use warnings;
use One;
sub one { One::one(@_) }
sub two (&) {
 join ', ', $_[0]->()
}
1;
```

Question 7

```
$ cat One.pm
package One;
use strict; use warnings;
use Two;
sub one {
 Two::two {
  (foo => 'bar', baz => 'quux')
 }
}
1;
```

```
$ cat Two.pm
package Two;
use strict; use warnings;
use One;
sub one { One::one(@_) }
sub two (&) {
 join ', ', $_[0]->()
}
1;
```

```
$ perl -MOne -E 'say One::one()'
```

Question 7

```
$ cat One.pm
package One;
use strict; use warnings;
use Two;
sub one {
 Two::two {
  (foo => 'bar', baz => 'quux')
 }
}
1;
```

```
$ cat Two.pm
package Two;
use strict; use warnings;
use One;
sub one { One::one(@_) }
sub two (&) {
 join ', ', $_[0]->()
}
1;
```

```
$ perl -MOne -E 'say One::one()'
foo, bar, baz, quux
```

Question 7

```
$ cat One.pm
package One;
use strict; use warnings;
use Two;
sub one {
 Two::two {
  (foo => 'bar', baz => 'quux')
 }
}
1;
```

```
$ cat Two.pm
package Two;
use strict; use warnings;
use One;
sub one { One::one(@_) }
sub two (&) {
 join ', ', $_[0]->()
}
1;
```

```
$ perl -MTwo -E 'say One::one()'
```

Question 7

```
$ cat One.pm
package One;
use strict; use warnings;
use Two;
sub one {
 Two::two {
  (foo => 'bar', baz => 'quux')
 }
}
1;
```

```
$ cat Two.pm
package Two;
use strict; use warnings;
use One;
sub one { One::one(@_) }
sub two (&) {
 join ', ', $_[0]->()
}
1;
```

```
$ perl -MTwo -E 'say One::one()'
Can't use string ("foo") as a subroutine ref while "strict refs"
in use at Two.pm line 6.
```

Question 8

```
$ cat q8.pl
use 5.010; use strict; use warnings;
sub F { 'G' }
my $f = F->new;

$ perl q8.pl
```

Question 8

```
$ cat q8.pl
use 5.010; use strict; use warnings;
sub F { 'G' }
my $f = F->new;

$ perl q8.pl
Can't locate object method "new" via package "G"
(perhaps you forgot to load "G"?) at q8.pl line 4.
```

Question 8

```
$ cat q8.pl
use 5.010; use strict; use warnings;
sub F { 'G' }
my $f = F::->new;

$ perl q8.pl
Can't locate object method "new" via package "F"
(perhaps you forgot to load "F"?) at q8.pl line 4.
```

Question 8

```
$ cat q8.pl
use 5.010; use strict; use warnings;
use autodie; open F, '<', '/dev/null';
my $f = F::->new;

$ perl q8.pl
```

Question 8

```
$ cat q8.pl
use 5.010; use strict; use warnings;
use autodie; open F, '<', '/dev/null';
my $f = F::->new;

$ perl q8.pl
Can't locate object method "new" via package "IO::File"
at q8.pl line 4.
```

En résumé

`new X` compile en un appel de méthode lorsque :

- soit la sub `new` n'a pas encore été déclarée dans le package courant ;
- soit elle l'est, mais `X` n'est pas un package vide.

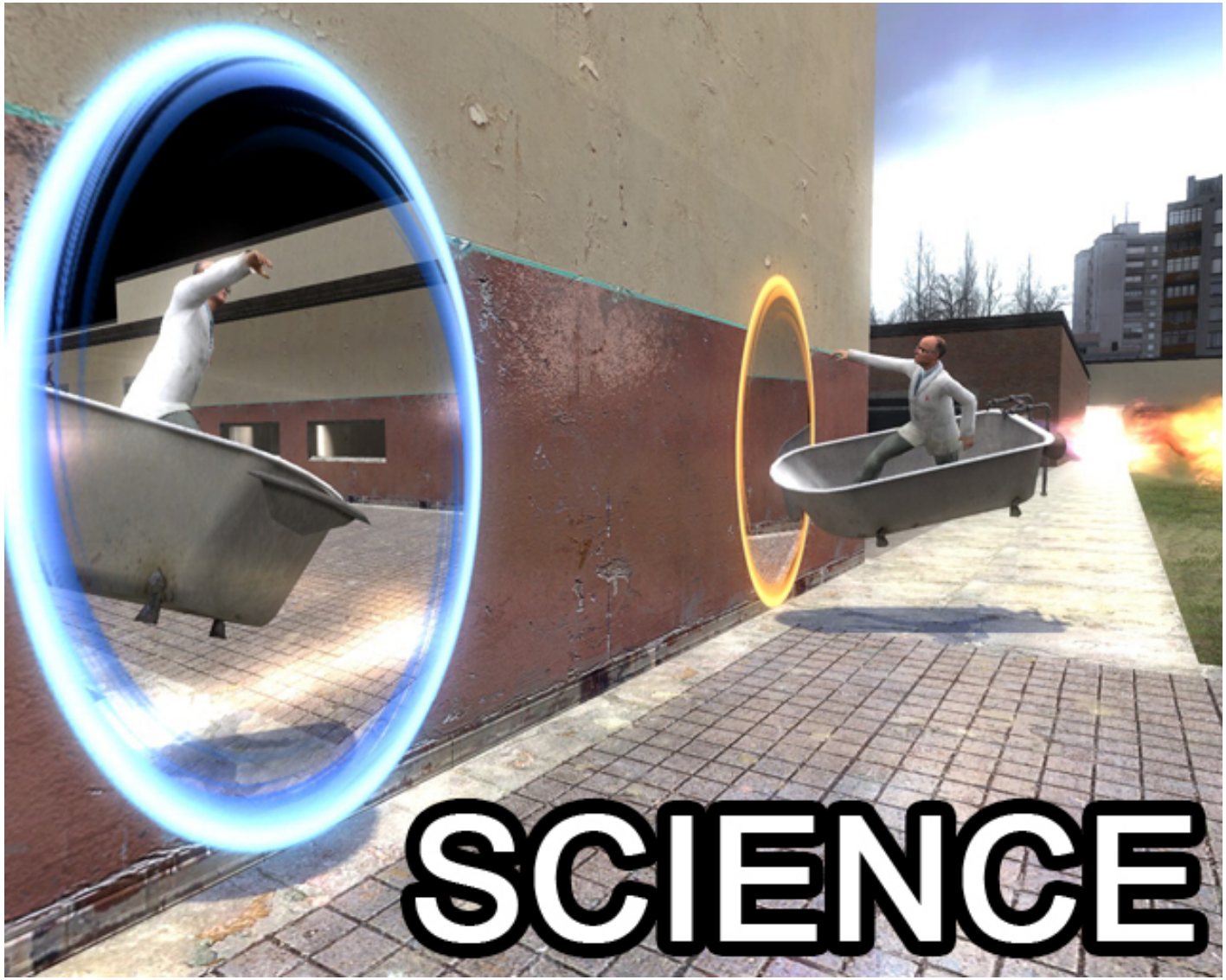
En résumé

`new X` compile en un appel de méthode lorsque :

- soit la sub `new` n'a pas encore été déclarée dans le package courant ;
- soit elle l'est, mais `X` n'est pas un package vide.

`X->new` signifie toujours un appel de méthode, mais attention à la classe.

Comment dire non ?



Comment dire non ?

indirect

```
$ cat indirect1.pl
#!perl
use strict; use warnings;
my $x = new X;

$ perl -c indirect1.pl ; echo $?

indirect1.pl syntax OK
0
```

Comment dire non ?

indirect

```
$ cat indirect1.pl
#!perl
use strict; use warnings;
no indirect;
my $x = new X;

$ perl -c indirect1.pl ; echo $?
Indirect call of method "new" on object "X"
at indirect1.pl line 4.
indirect1.pl syntax OK
0
```

Comment dire non ?

indirect

```
$ cat indirect1.pl
#!perl
use strict; use warnings;
no indirect;
my $x = new $class;

$ perl -c indirect1.pl ; echo $?
Indirect call of method "new" on object "$class"
at indirect1.pl line 4.
indirect1.pl syntax OK
0
```

Comment dire non ?

indirect

```
$ cat indirect1.pl
#!perl
use strict; use warnings;
no indirect;
my $x = new { foo => 1 };

$ perl -c indirect1.pl ; echo $?
Indirect call of method "new" on a block
at indirect1.pl line 4.
indirect1.pl syntax OK
0
```

Comment dire non ?

indirect

```
$ cat indirect1.pl
#!perl
use strict; use warnings;
no indirect ':fatal';
my $x = new { foo => 1 };

$ perl -c indirect1.pl ; echo $?
Indirect call of method "new" on a block
at indirect1.pl line 4.
255
```

Comment dire non ?

indirect

```
$ cat indirect2.pl
#!perl
use strict; use warnings;
my $x = new X;
no indirect;
my $y = new Y;
foo();
{
 my $z = new Z;
 use indirect;
 my $t = new T;
}
my $u = new U;

$ perl -c indirect2.pl
Indirect call of method "new" on object "Y" at indirect2.pl line 5.
Indirect call of method "new" on object "Z" at indirect2.pl line 8.
Indirect call of method "new" on object "U" at indirect2.pl line 12.
indirect2.pl syntax OK
```

Comment dire non ?

`indirect`

- s'applique uniquement au bloc lexical courant ;
- fonctionne à partir de perl 5.8.1 ;
- ne s'applique pas à `print` et compagnie (c'est voulu) ;
- la variable d'environnement `PERL_INDIRECT_PM_DISABLE` permet de désactiver le module en production.

Comment dire non ?

`Perl::Critic::Policy::Dynamic::NoIndirect`

```
$ cat x.pl
new X;
$ perlcritic -s \
  Perl::Critic::Policy::Dynamic::NoIndirect x.pl
Indirect call of method "new" on object "X" at line 1,
column 1.  You really wanted object "X"->new.
```

- analyse dynamique, compile le code.
- rajoute `no indirect` au début du code, compile, puis retranscrit les messages de `indirect` en erreurs `Perl::Critic`.

Comment dire non ?

`Perl::Critic::Policy::Objects::ProhibitIndirectSyntax`

- analyse statique, ne compile pas le code ;
- les méthodes interdites doivent être spécifiées dans le fichier de configuration de `Perl::Critic` ;
- peut donner des faux positifs et des faux négatifs.