Far More Than You Ever Wanted To Know

about

Typeglobs, Closures and Namespaces

Typeglobs, Closures e Namespace

Daniel Ruoso
daniel@ruoso.com

Em um curso de introdução...

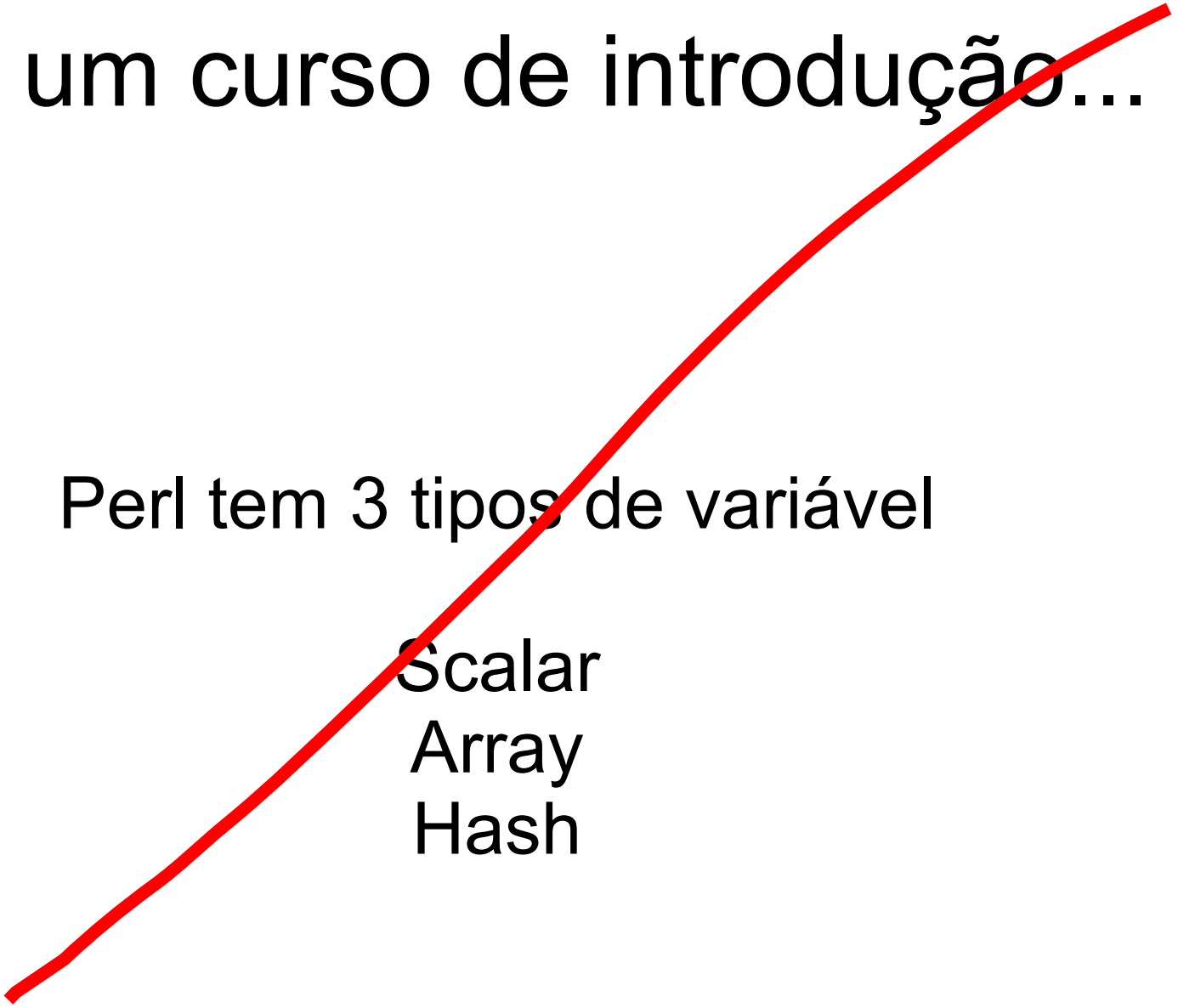
Perl tem 3 tipos de variável

Scalar
Array
Hash

Em um curso de introdução...

Perl tem 3 tipos de variável

Scalar
Array
Hash



Em um curso de introdução...

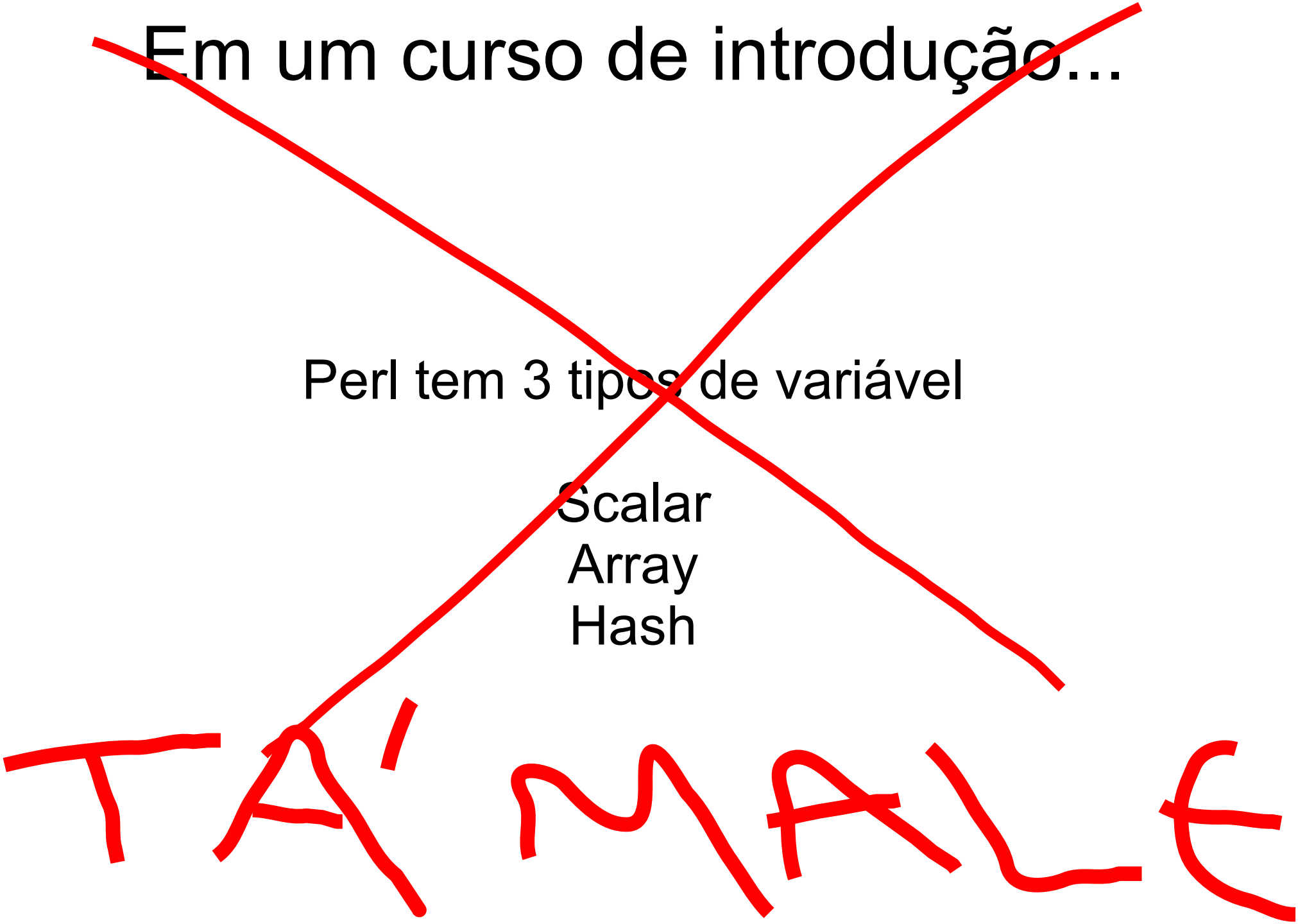
Perl tem 3 tipos de variável

Scalar
Array
Hash

Em um curso de introdução...

Perl tem 3 tipos de variável

Scalar
Array
Hash



São 7 tipos de **VARIÁVEIS**

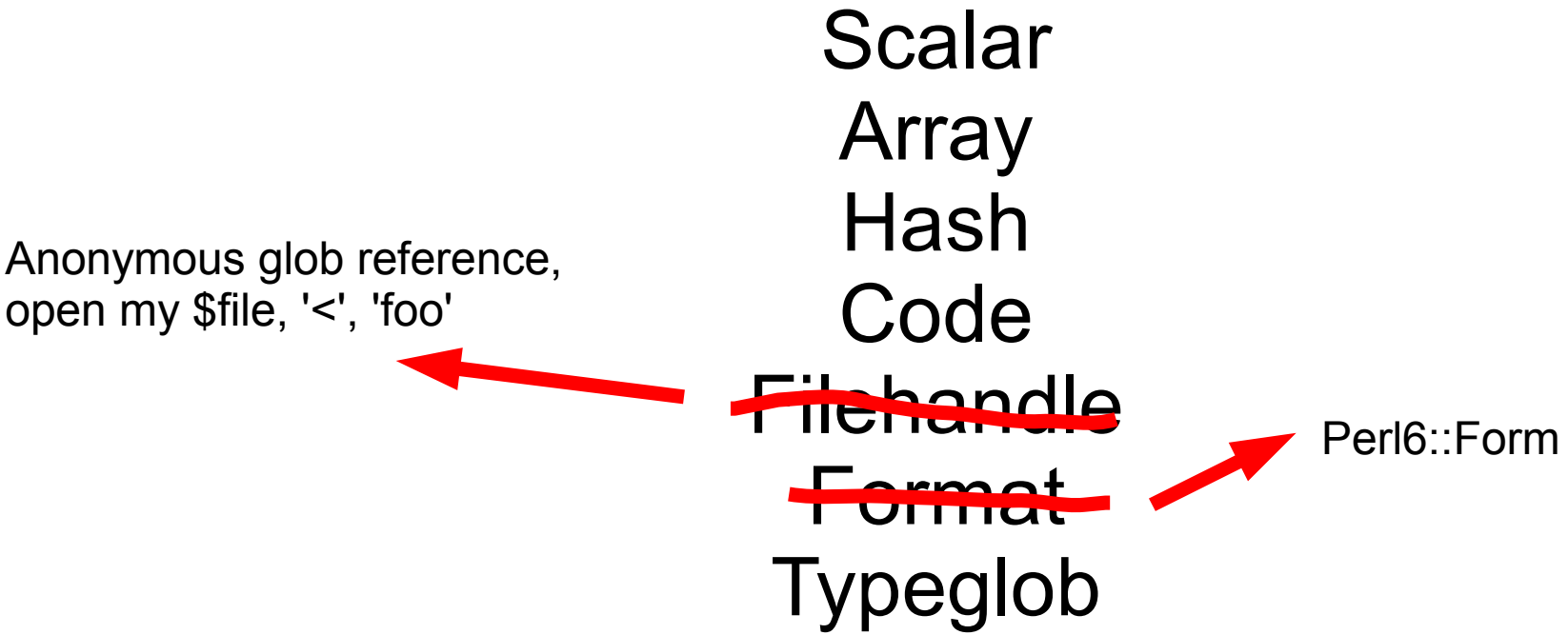
Scalar
Array
Hash
Code
Filehandle ou simplemente IO
Format
Typeglob

São ~~7~~ 6 tipos de **VARIÁVEIS**

Scalar
Array
Hash
Code
Filehandle
~~Format~~
Typeglob

Perl6::Form

São 7 tipos de **VARIÁVEIS**



São 7 tipos de **VARIÁVEIS**

Scalar - $foo
Array - @foo
Hash - %foo
Code - &foo
~~Filehandle~~
~~Format~~
Typeglob - *foo

Anonymous glob reference,
open my $file, '<', 'foo'

Perl6::Form

São 7 tipos de **VALORES**

Scalar
Array
Hash
Code
Filehandle
Format
Typeglob

São 7 tipos de **VALORES**

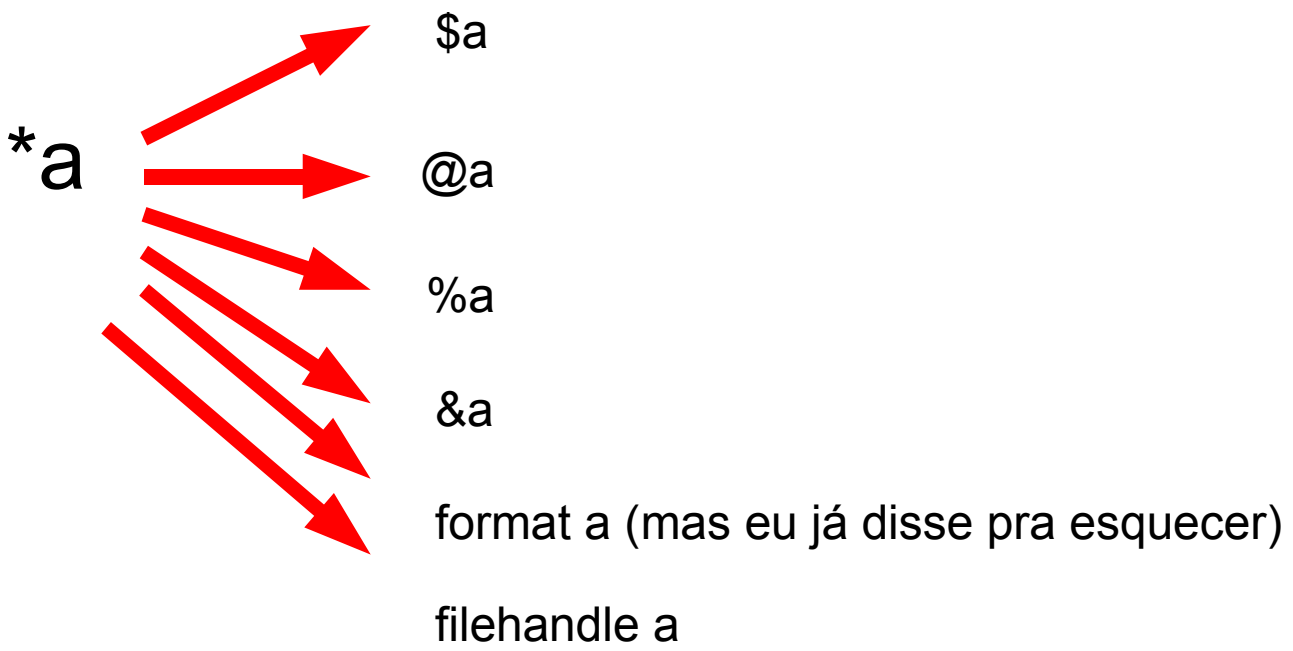
Scalar - $foo = 1
Array - @foo = (1,2,3)
Hash - %foo = (a => 1, b => 2)
Code – sub { ... }
Filehandle – open my $file
Format – (esqueçam que este existe)
Typeglob - ...

Typeglob

De maneira simplificada:
“Um typeglob é um Símbolo”

De maneira extensa:
“Typeglob é onde todos os possíveis tipos de valores de um símbolo são armazenados”

Typeglob



Typeglob

Isso significa que

open my $file, '<', 'foo'

Namespace

Toda variável global está no namespace do interpretador, dentro do ::main.

::main é um hash

onde os nomes apontam para typeglobs

$::a = ${*main::a} = ${“main::a”} = ${$main::{a}}

Namespace

${*main::a}

Compile-time

Mais eficiente

Namespace

${“main::a”}

Interpolação de Strings

FEIO

precisa de “no strict 'refs'”

Namespace

${$main::{a}}

Runtime

BONITO

funciona com “use strict”

Namespace

```
use strict;
$::a = 1;
my $b = 'a';
print ${$main::{$b}};
```

Permite lookup dinâmico

Criando uma named subroutine em runtime usando uma anon sub

```
use strict;
my $name = 'hello';
my $code = sub { 'Hello '.$_[0].'!' };
$main::{$name} = $code;
print hello('World');
```

Closures

Uma closure é um pedaço de código associado a um pedaço de memória.

É quase como um objecto, mas não pertence a nenhuma classe. Ele mesmo é o código.

Closures

```
sub gen_greeter {
   my $greeter = shift;
   return sub {
      my $what = shift;
      return $greeter.' '.$what.'!';
   };
};

my $hello = gen_greeter('hello');
my $howdy = gen_greeter('howdy');
print $hello->('World');
print $howdy->('World');
```

Closures

Mas uma closure ainda é simplesmente um CODE.

print ref $hello; # CODE

Closures

Sendo assim, ela pode estar dentro de um typeglob...

```
*::main::hello = $hello;
print hello('world');
```

Criando uma Classe dinamicamente

Para isso, vamos usar closures e manipulação de typeglobs.

Criando uma Classe dinamicamente

o nosso código de criar uma classe será:

ClassFactory->create('Foo::Bar', 'a', 'b', 'c')

'Foo::Bar' é o nome da classe
a, b e c são atributos que terão um accessor

Criando uma Classe dinamicamente

```
package ClassFactory;

use strict;

sub create {
        my ($self, $class, @attr) = @_;
        no strict 'refs';
        *{$class.'::new'} = sub {
                my $class = shift;
                $class = ref $class || $class;
                my %args = @_;
                return bless \%args, $class;
        };
        for my $attr (@attr) {
                *{$class.'::'.$attr} = sub {
                        my ($self, $newval) = @_;
                        $self->{$attr} = $newval if ($newval);
                        return $self->{$attr};
                };
        };
        return $class;
}

1;
```

```
 # classfactory.pl
ClassFactory->create(qw(Foo::Bar a b c));
my $o = Foo::Bar->new(a => 1, b => 2, c => 3);
$o->a(5);
print join ', ', $o->a, $o->b, $o->c;
print $/;
```

```
$ perl classfactory.pl
5, 2, 3
```

O Hack Final

TXTClassLoader

Procura no @INC um arquivo .txt ao invés de .pm e gera a classe com os atributos que estejam listados, um por linha, naquele arquivo

TXTClassLoader

```
#MyClass.txt:
a
b
c

#test.pl:
use TXTClassLoader;
use MyClass;

my $obj = MyClass->new(a => 1,
               b => 2, c => 3);
$obj->a(5);
print join ', ', $obj->a, $obj->b, $obj->c;
print $/;
```

```
#running
$ perl test.pl
5, 2, 3
```

```
#TXTClassLoader.pm
package TXTClassLoader;
use ClassFactory;
push @INC, sub {
    my ($self, $class) = @_;
    my $file = $class;
    $file =~ s/\.pm$/.txt/;
    $class =~ s/\.pm$//;
    $class =~ s/\//::/g;
    my ($path) = grep { !ref($_) && -e $_.'/'.$file } @INC;
    if ($path) {
        open my $fh, '<', $path.'/'.$file || die $!;
        my @attr = map { chomp; $_ } <$fh>;
        close $fh;
        ClassFactory->create($class,@attr);
        my $returncode = '1;';
        open my $ret, '<:scalar', \$returncode;
        return $ret;
    } else {
        return 0;
    }
};
1;
```

Alguém ainda está prestando atenção?

Perguntas?