Design by Contract в Perl

Вячеслав Тихановский (vti)

2012-12-22

showmetheco.de

ООП

Moose, Moos, Moo, Mo, M?

Интерфейсы и наследование от абстрактного класса

```
package AbstractRenderer;

sub new {...}

sub render { die 'Implement␣me!' }
```

```
package TT;
use base 'AbstractRenderer';
sub render {
    my $self = shift;
    my ($template, $layout, %vars) = @_;

    return \$rendered;
}

package Caml;
use base 'AbstractRenderer';
sub render {
    my $self = shift;
    my ($template, %vars) = @_;

    return $rendered;
}
```

Design by Contract

- Проверка входных и выходных условий

- Проверка входных и выходных условий
- Следование принципу подстановки Лисков (L в SOLID) при наследовании

- Проверка входных и выходных условий
- Следование принципу подстановки Лисков (L в SOLID) при наследовании
- Документирование интерфейса

```
package AbstractRenderer;
sub new {...}

# Requires(STRING, %HASH_OF_ANY_VALUES)
# Ensures(STRING)
# Throws(Exception::TemplateNotFound)
sub render {
    my $self = shift;
    my ($template, %vars) = @_;

    ...

    return $rendered;
}
```

```
package TT;
use base 'AbstractRenderer';
sub render { ... }

package Caml;
use base 'AbstractRenderer';
sub render { ... }
```

- Не должно снижать читаемость

- Не должно снижать читаемость
- Не требует больших вычислительных ресурсов

- Не должно снижать читаемость
- Не требует больших вычислительных ресурсов
- Отключаемо

- Не должно снижать читаемость
- Не требует больших вычислительных ресурсов
- Отключаемо
- Наследуемо

- Не должно снижать читаемость
- Не требует больших вычислительных ресурсов
- Отключаемо
- Наследуемо
- Запрещает менять контракт в дочерних классах

- Не должно снижать читаемость
- Не требует больших вычислительных ресурсов
- Отключаемо
- Наследуемо
- Запрещает менять контракт в дочерних классах
- Понятные ошибки

- Не должно снижать читаемость
- Не требует больших вычислительных ресурсов
- Отключаемо
- Наследуемо
- Запрещает менять контракт в дочерних классах
- Понятные ошибки
- Как можно меньше магии*

- Не должно снижать читаемость
- Не требует больших вычислительных ресурсов
- Отключаемо
- Наследуемо
- Запрещает менять контракт в дочерних классах
- Понятные ошибки
- Как можно меньше магии*
- Похоже на Perl*

Class::Contract

```
use Class::Contract;

contract {
  inherits 'BaseClass';

  ctor 'new';

  method 'methodname';
    pre  { ... };
      failmsg 'Error message';

    post  { ... };
      failmsg 'Error message';

    impl { ... };
};
```

Sub::Contract

```
use Sub::Contract qw(contract);

contract('surface')
    ->in(\&is_integer, \&is_integer)
    ->out(\&is_integer)
    ->enable;

sub surface {
    # no need to validate arguments anymore!
    # just implement the logic:
    return $_[0] * $_[1];
}
```

Sub::Assert

```
assert
      pre      => {
        'parameter larger than one' => '$PARAM[0] >
      },
      post     => '$VOID or $RETURN <= $PARAM[0]',
      sub      => 'squareroot',
      context  => 'novoid',
      action   => 'carp';
```

Carp::Datum

```
use Carp::Datum;

sub routine {
    DFEATURE my $f_, "optional␣message";
    my ($a, $b) = @_;
    DREQUIRE $a > $b, "a␣>␣b";
    $a += 1; $b += 1;
    DASSERT $a > $b, "ordering␣a␣>␣b␣preserved";
    my $result = $b - $a;
    DENSURE $result < 0;
    return DVAL $result;
}
```

MooseX::Contract

```
contract 'add'
    => accepts [ ...type... ]
    => returns void,
    with_context(
        pre => sub {
            ...
        },
        post => assert {
            ...
        }
    );
sub add {
    my $self = shift;
    my $incr = shift;
    $self->{value} += $incr;
    return;
}
```

Attribute::Contract

```
package AbstractRenderer;
use Attribute::Contract;

sub new {...}

sub render
   :ContractRequires(VALUE(Str), %ANY)
   :ContractEnsures(VALUE(Str))
   :ContractThrows(Exception::TemplateNotFound) {
}
```

DbC и TDD

DbC и Defensive programming

Когда применять?

- Библиотеки, модули
- Интерфейсы

Как не применять?

- Не валидировать данные
- Не производить побочных эффектов

Спасибо!