

C o n s t r u c t i v e D e s t r u c t o r U s e i n O b j e c t - O r i e n t e d P e r l

T e r r e n c e B r a n n o n

<http://www.metaperl.org>

metaperl@gmail.com

Vanilla Perl OO Destructors

a class with destructor

```
package X;
use strict; use warnings;

sub new {
    bless {}, __PACKAGE__;
}

sub DESTROY {
    my($self)=@_;
    print "$self is dead\n";
}
```

program and output

```
print "before birth\n";
{
    my $x = X->new;
}
print "post-mortem\n";
```

```
before birth
X=HASH(0x10023ce8) is dead
post-mortem
```

Moose Destructors

program and output

```
package X;
use Moose;

sub DEMOLISH {
    my($self)=@_;
    print "$self is dead\n";
}
```

```
print "before birth\n";
{
    my $x = X->new;
}
print "post-mortem\n";
```

```
before birth
X=HASH(0x10654b68) is dead
post-mortem
```

S o F a r..

S o w h a t ? !

J u s t a b i t m o r e b a c k g r o u n d ...

X M L :: W r i t e r

Code and output

```
$writer->startTag("level1");  
$writer->startTag("level2");  
$writer->startTag("level3");  
$writer->endTag();  
$writer->endTag();  
$writer->endTag();
```

```
<level1>  
  <level2>  
    <level3></level3>  
  </level2>  
</level1>
```

Code critique

- have to manually synchronize start and end tags or invalid XML *will* be produced
- the Perl code is not nested or indented (not suggestive of the XML structure)

XML::Writer has no API support for element nesting

```
use XML::Writer;  
  
my $writer = XML::Writer->new;  
  
$writer->dataElement("level1", 45);
```

```
<level1>  
  45  
</level1>
```

XML::Writer::Nest – Phase 1

```
my $writer = new XML::Writer;
```

```
my $level1 = XML::Writer::Nest->new (tag => 'level1', writer => $writer );
```

```
my $level2 = $level1->nest ('level2');
```

```
my $level3 = $level1->nest('level3');
```

- open/close tags automatically match!
- indentation *still not* suggestive of XML structure
- reference decrement in LIFO order is current (but not promised) implementation – http://www.perlmonks.org/?node_id=812228

XML::Writer::Nest

Phase 2

```
my $writer = new XML::Writer;
```

```
{ my $level1 = XML::Writer::Nest->new(tag => 'level1', writer => $writer );
```

```
  { my $level2 = $level1-> nest ('level2');
```

```
    {
```

```
      my $level3 = $level2->nest('level3');
```

```
    }
```

```
  }
```

```
}
```

- open/close tags automatically match!
- indentation suggestive of XML structure
- no worries about undocumented change in implementation

if-then, nesting, loops

```
if (condition_A) {  
  my $a = $xml->nest('A_true');  
  {  
    my $a2 = do {  
      if (condition_A2) {  
        $xml->nest('a2_true');  
      } else {  
        $xml->nest('a2_false');  
      }  
    };  
  }  
} else {  
  for my $i (@i) {  
    my $tag = $xml->nest( "sibling_$i", attr_num => $i )  
  }  
}
```

m o r e s t a c k , l e s s v a r s

```
use XML::Writer::Nest;
```

```
my $writer = new XML::Writer;
```

```
{ local $xml = XML::Writer::Nest->new(tag => 'level1', writer => $writer );
```

```
    { local $xml = $xml-> nest ('level2');
```

```
        {
```

```
            local $xml = $xml->nest('level3');
```

```
        }
```

```
    }
```

```
}
```

Implementation of XML::Writer::Nest

```
package XML::Writer::Nest;
```

```
use Moose;
```

```
has 'tag' => (isa => 'Str', is => 'ro', required => 1);
```

```
has 'attr' => (
```

```
    isa => 'ArrayRef[Maybe[Str]]',
```

```
    is => 'ro',
```

```
    default => sub { [] }
```

```
);
```

```
has 'writer' => (isa => 'XML::Writer', is => 'ro', required => 1);
```

Implementation of XML::Writer::Nest

```
sub BUILD {  
    my($self)=@_  
  
    my @attr = defined($self->attr) ? @{$self->attr} : ();  
    $self->writer->startTag($self->tag, @attr);  
}  
  
sub DEMOLISH {  
    my($self)=@_  
    $self->writer->endTag();  
}
```

Related XML Generation Work

- XML::Generator, XML::Toolkit (Moose <=> XML)
- “A data structure for XML Generation”

http://perlmomks.org/?node_id=787605

```
my $h = HTML::Element->new_from_lol
```

```
(  
  [ level1 =>  
    [ level2 =>  
      [ 'level3' ]  
    ],  
  ]  
);
```

```
warn $h->as_HTML('<>&', "\n\t"); # http://pastie.org/pastes/738006
```

F o r m o r e c o m p l e x e x a m p l e s . . .

HTML::Element::Replacer is a more
complex example of "constructive
destruction" used to simplify use of
HTML::TreeBuilder

<<http://search.cpan.org/dist/HTML-Element-Replacer/>>