Using Moose to build an Object Oriented Application
### Moose Quick-Ref Card

**A modern object system for Perl 5**

#### Exported Functions

- `use Moose;`  
  Turns on strict and warnings. Exports `confess` and `blessed`.

- `extends @superclasses`  
  Moose’s alternative to use `base`. Note that it will re-set `@ISA`.

- `with @roles`  
  `with $role => { %options }`  
  Consume roles (interfaces) as an alternative to extending classes.

- `has $name => { %options }`  
  Install an attribute into this class. See below for `%options` details.

- `has "+{$name}" => { %options }`  
  Clone and extend an attribute.

- `before $names => sub {...}`  
  `around $names => sub {...}`  
  `after $names => sub {...}`  
  Extend a superclass’s method. around is passed (\$coderef, \$self, \$args).

- `override $name => sub { super() }`  
  Explicit override of a method.

- `augment $name => sub { inner() }`  
  The inverse of override/super.

- `dump`  
  Output object using `Data::Dumper`.

#### Attribute Constructor Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`is =&gt; 'rw'</td>
<td>'ro'`</td>
</tr>
<tr>
<td>`isa =&gt; $type_name</td>
<td>'$ta</td>
</tr>
<tr>
<td><code>does =&gt; $role</code></td>
<td>Value’s class must consume <code>$role</code>.</td>
</tr>
<tr>
<td><code>metaclass =&gt; $name</code></td>
<td>Extend attribute via a metaclass.</td>
</tr>
<tr>
<td><code>traits =&gt; [@role_names]</code></td>
<td>Apply roles to attribute’s meta-object.</td>
</tr>
<tr>
<td>`coerce =&gt; 1</td>
<td>0`</td>
</tr>
<tr>
<td>`required =&gt; 1</td>
<td>0`</td>
</tr>
<tr>
<td>`weak_ref =&gt; 1</td>
<td>0`</td>
</tr>
<tr>
<td>`lazy =&gt; 1</td>
<td>0`</td>
</tr>
<tr>
<td>`auto_deref =&gt; 1</td>
<td>0`</td>
</tr>
</tbody>
</table>

#### Additional Options

- `trigger => sub {...}`  
  Code to run after attribute is set. Is passed (`$self`, `$new_val`).

- `default => $val | sub { [] | {} | {sub {...} }`  
  Default value to initialize attribute. The outer `sub()` is passed `$self`.

- `predicate => $name`  
  Method `$name` will perform a basic defined test on the attribute.

- `reader|writer|clearer => $sub_name`  
  Provide your own subroutines to read from, write to, and uninitialized the stored value.

- `builder => $sub_name`  
  Separate method to return default value. Better for subclassing.

- `lazy_build => 1`  
  Sets lazy, required, predicate (has $name), clear (clear $name) and builder (_build $name).

- `accessor => sub {...}`  
  Provide your own read/write accessor.

- `init_arg => $name`  
  Name for attribute when passed into the constructor, or disallowed if `undef`.

- `handles => @ary@hash|qr//|$role|sub(...)`  
  Sets up methods which delegate to methods of the value’s class. Requires that `isa` be set.
**Data Type Constraints**

The built-in type-constraints are:

- **Any**
- **Item**
- **Bool**
- **Undefined (use with care)**
- **Defined**
- **Value**
  - **Num**
  - **Str**
    - **ClassName** (means “is loaded” and isa)
- **Ref**
  - **ScalarRef**
  - **ArrayRef or ArrayRef[TypeName]**
  - **HashRef or HashRef[TypeName]**
  - **CodeRef**
  - **RegExpRef**
  - **GlobRef**
  - **FileHandle**
  - **Object**
  - **Role**

To define your own, global types:

```perl
use Moose::Util::TypeConstraints;

type $name
  => where { <code> }
  => message { $message };

A new type-constraint with no parent.
```

```perl
subtype $name
  => as $Parent
  => where { <code> }
  => message { $message };

Subtype of an existing type.
```

It is recommended that you always quote $name. Moose checks $parent constraints first. The block of <code> must evaluate to true. A $message is optional, and used in confess if the constraint check fails.

**Data Type Coercions**

```perl
use Moose::Util::TypeConstraints;

coeerce $type
  => from $some_type
  => via { <code> }
  => from $some_other_type
  => via { <other_code> };

Instruct Moose in how to coerce data from $some_type to $type. You can chain alternative coercions as shown.
```

Coercion <code> is passed a value in $_ and returns the value to be stored.

**Other Tidbits**

```perl
use Moose::Role;

A role (or interface or trait) can only be consumed, not instantiated directly.
```

```perl
requires @methods;

Methods which must be implemented by the consuming class.
```

```perl
my $meta = __PACKAGE__->meta;

Get the cached metaclass for a package.
```

```perl
$meta->make_immutable;

no Moose;

no Moose::Role;

Finalize the class to make it faster, and unimport the Moose 'keywords'.
```

The BUILD method of each class will be executed after the type constraint checks by the constructor, and is passed ($self, $params).

Before that, BUILDARGS is passed @params to convert into the $params hashref.

The DEMOLISH method of each class is called at object destruction.

**Meta Class and Trait namespaces:**

Moose::Meta::Attribute::Custom::MetaClass
Moose::Meta::Type::Custom::Trait::Trait

This quick-ref card is © Oliver Gorwits
2009-01-07 version 3.8
http://tinyurl.com/moosequickref
Thanks to many people from #moose
The goal of Moose is to increase programmer productivity by providing “proper” OO syntax and Object model which are missing from perl5. It does the grunt work for you and allows the programmer to concentrate on the task at hand.

What is Moose?
An Example

A “Traditional” OO Perl Class

Package ClassName;
sub new {
    my $class = shift;
    my $self = {
        _an_attribute => shift,
        _another_attribute => shift,
    };
    bless $self, $class;
}

package Point;
use Moose;
has 'x' => (isa => 'Int', is => 'rw', required => 1);
has 'y' => (isa => 'Int', is => 'rw', required => 1);
sub clear {
    my $self = shift;
    $self->x(0);
    $self->y(0);
}

package Point3D;
use Moose;
extends 'Point';
has 'z' => (isa => 'Int', is => 'rw', required => 1);

after 'clear' => sub {
    my $self = shift;
    $self->z(0);
};

package main;
my $point1 = Point->new(x => 5, y => 7);
my $point2 = Point->new({x => 5, y => 7});
my $point3d = Point3D->new(x => 5, y => 42, z => -5);
package Document::Page;
use Moose;
has 'body' => ( is => 'rw', isa => 'Str', default => sub {''})
sub create {
    my $self = shift;
    $self->open_page;
    inner();
    $self->close_page;
}
sub append_body {
    my ( $self, $appendage ) = @_; 
    $self->body( $self->body . $appendage );
}
sub open_page  { (shift)->append_body('<page>') }
sub close_page { (shift)->append_body('</page>') }

package Document::PageWithHeadersAndFooters;
use Moose;
extends 'Document::Page';
augment 'create' => sub {
    my $self = shift;
    $self->create_header;
    inner();
    $self->create_footer;
};
sub create_header { (shift)->append_body('<header/>') }
sub create_footer { (shift)->append_body('<footer/>') }

package Point;
use Moose;
use namespace::clean -except => 'meta';

has 'x' => ( isa => 'Int', is => 'ro' );
has 'y' => ( isa => 'Int', is => 'rw' );

__PACKAGE__->meta->make_immutable;
use Moose;
    extends 'A::Base::Class';
with qw(
    DoesSomething::Well
    DoesSomething::Else
    DoesSomething::Difficult
);

package Breakable;
use Moose::Role;
has 'is_broken' => (
    is => 'rw',
    isa => 'Bool',
);
sub break { my $self = shift;
    $self->is_broken(1);
}

package Car;
use Moose;
with 'Breakable';

has 'engine' => (
    is => 'ro',
    isa => 'Engine',
);