Object Oriented Perl: An Introduction

Abram Hindle

Department of Computer Science

University of Victoria

abez@uvic.ca

July 13, 2004

This Presentation

- What am I going to cover?
  - OO Intro
  - Packages
  - References
  - Constructor
  - Attributes
  - Methods
  - Inheritance
  - Fun Stuff
  - Getting Help
  - Conclusions
  - References

OO Introduction

- What are objects?
  - A combination of attributes and functions associated with those attributes. Objects can have state, identify and behaviors. You can execute actions associated with objects. e.g. A dog object could bark. Effectively objects are instances of Classes.
- What are classes?
  - The definition or blueprint for an object.

    http://www.justpbinfo.com/techdocs/ooterms.asp

OO Introduction

- Why use Object Oriented Programming
  - OOP helps us organize our code and split our problems up into smaller blocks.
  - OOP is the current dominant programming style.
  - OOP seems to be the best that we currently have in imperative programming.
  - With OOP we gain useful things like polymorphism, encapsulation and other big words/concepts.

Packages

- What are Packages?
  - The main way code is organized in Perl is with packages.
  - Sample Package:

```
* package Name;
  use strict; #save us debugging time
  ...code...
  1; # outdated, used for a return from a use.
```

  - More than 1 package can exist in a file. Usually packages are saved into .pm files like Name.pm

Packages

- More Details...
  - Our Class Names will be the package name.
  - Packages can be included by using them:
    - `use Name;`
  - If a package in a subdirectory attach {*dirname*}*::*{*packagename*} .
    - `use SubDir::Name;`
    - `package SubDir::Name;`

Constructor

- What is a blessing?
  - Blessing is taking a scalar and associating it with a class thus making a reference, an object.
  - Perl Classes can be any Perl data type:
  - Scalar,Array,Hash,glob,etc. as long as you make a reference to it.
  - HashRefs are the easiest to use.
  - We can make a reference become a class (that we've defined) by blessing it!
    - `bless($hashRef,"Name");`

Constructor

- How do constructors work in Perl?
  - The function ”new” has special properties in Perl:
    - `$obj = Name->new();`
    - `$obj = new Name();  ]`
  - We can make a constructor like so:

```
sub new {
        my $type = shift;
        $type = ref($type) || $type;
        my $self = {};
        bless($self,$type);
        return $self;
}
```

Attributes

- How do attributes work in Perl?
  - Since our object is a hashref we can store dynamic information inside of our object. We can only store scalars, so everything is scalar or a reference.

```
* $self->{counter} = 1;
  $self->{array} = \@arr;
  $self->{hash} = \%hash;
  $arrayref = [@array];
  $hashref = {%hash};
```

  - We can have class variables (static) by declaring the variable global in the package:

```
* my $counter = 0;
```

Attributes

- What about private attributes?
  - The quick answer to ”what about private data members?” is No, you don’t have private attributes. In Perl everything is public unless you try really really hard then you can add further layers of encapsulation.
  - Everything is PUBLIC, everyone can access your objects they way you don’t want them to. Be responsible when coding to stay safe and use defined interfaces.

Methods

- What about methods in Perl?
  - Methods are functions which are associated with a class. We declare methods inside of the package file containing our class.
  -
```
sub get {#generic getter
        my ($self,$attr) = @_;
        return $self->{$attr};
}
sub set {#generic setter
        my ($self,$attr,$value) = @_;
        $self->{$attr} = $value;
}
sub inc {
        my $self = shift;
        return ++$self->{count};
}
```

Methods

- How Do I call a method?
  - To call a method you reference the object that had the method with $->$ operator `Obj->method` .
  - ```
    my $obj = new Name();
    $obj->method();
    $obj->method;
    Name->staticMethod();
    ```
    ```

Methods

- Method names can come from scalars
  -
```
my $methodname = "getName";
print Name->new()->$methodname(),"\n";

#Alternatively
#You can reference methods with strings really easy. Great for CGI
     if you have a "safety hash"
@methods = qw(method1 method2 method3);
%safe{method1} = 1;
%safe{method2} = 1;
%safe{method3} = 1;
$method = $methods[rand(scalar(@methods))];
if ($safe->{$method}) { $obj->$method(); }
```

Destructors

- How does Perl handle destructors?
  - Destructors are called when Perl garbage collection finds an instance of a class to destroy.
  - Perl does have destructors although they aren't too useful. Sometimes you might be abstracting a file handle so a destructor would be useful.
  - ```
    sub DESTROY {
            my $self = shift;
    }
    ```
    ```

Inheritance

- How does Perl and Inheritance Work?
  - Inheritance allows classes to ”inherit” or gain the functionality of a super class but possibly overload some of the functions. A Sub class IS A member of a super class.
    - Dog and Cat inherit from Mammal. A dog IS A Mammal, a Cat is not a Dog and a Mammal is not a Dog or a Cat. Mammals are warm blooded and furry, dogs and cats are too.

Inheritance

- cont..
  - To inherit from a super class in Perl we ”use base” in the sub class.
  - `use base "SuperClass" ;`
  - `use base qw(SuperClass); #must provide a string`
  - Objects ”using base” now can call all the methods of the super object. Static methods are not guaranteed to work properly. Static attributes will not work.
  - Perl supports multiple inheritance.

Inheritance

- cont..
  - Every Perl class inherits methods from UNIVERSAL.
    `$obj->isa("Type")` , is a common method used.
  - The package SUPER refers to the super class.
  - E.g. a inherited constructor could be:
    * `sub new { return shift()->SUPER::new(@_); }`

Inheritance

```
package Mammal; #Mammal.pm
sub new {
        my $type = ref($_[0]) || $_[0];
        my $self = {};
        bless($self,$type);
        return $self;
}
sub noise { return "FuzzFuzz"; }
1;
package Dog; #Dog.pm
use base qw(Mammal);
sub new { return shift()->SUPER::new(@_); }
sub noise { return "Bark"; }
1;
package Cat; #Cat.pm
use base qw(Mammal);
sub new { return shift()->SUPER::new(@_); }
sub noise { return "Meow"; }
```

```
1;
```

Inheritance

- Using the Module
  - ```
    use Mammal;
    use Dog;
    use Cat;
    my $cat = new Cat();
    my $dog = new Dog();
    my $mam = new Mammal();
    foreach my $an ($cat,$dog,$mam) {
            print "The ".ref($an)." says ".$an->noise()."\n";
    }
    ```
    ```

Inheritance

- Output
  - `The Cat says Meow`
    `The Dog says Bark`
    `The Mammal says Fuzz Fuzz`

Autoloading

```
sub AUTOLOAD {
        return if $AUTOLOAD =~ /::DESTROY$/;
        no strict 'refs';
        if ($AUTOLOAD=~ /::get_(.*)$/) {
                my $attribute = $1;
                *{$AUTOLOAD} = sub { return shift()->{$attribute}; };
                goto &$AUTOLOAD;
        }
        if ($AUTOLOAD=~ /::set_(.*)$/) {
                my $attribute = $1;
                *{$AUTOLOAD} = sub {
                        my $self = shift;
                        $self->{$attribute} = shift;
                };
                goto &$AUTOLOAD;
        }
} # http://www.perl.com/lpt/a/2002/08/07/proxyobject.html
```

Get Help!

- Here are some sources where you can get help.
  - perldoc perltoot
  - perldoc perltootc
  - perldoc perlobj
  - Damian Conways's "Object Oriented Perl" book.

Conclusions

- Perl can easily be used in an Object Oriented Style
  - A Majority of Perl modules are done in a OO Style
  - Almost any OO trick you can do in any other language you can do in Perl
  - Perl usually is lacking in cases of control of attributes
  - Perl has a problem with colliding inherited attributes.