Object Oriented Programming in PERL

Before we start Object Oriented concept of perl, lets understand references and anonymous arrays and hashes

References

- A reference is, exactly as the name suggests, a reference or pointer to another object.
- There are two types of references: symbolic and hard.
- A symbolic reference enables you to refer to a variable by name, using the value of another variable. For example, if the variable $foo contains the string "bar", the symbolic reference to $foo refers to the variable $bar.
- A hard reference refers to the actual data contained in a data structure.

Creating Hard References

The unary backslash operator is used to create a reference to a named variable or subroutine, for example:

```
$foo = 'Bill';
$fooref = \$foo;
```

```
$fooref = \$foo;
```

The $fooref variable now contains a hard reference to the $foo variable. You can do the same with other variables:

```
$array = \@ARGV;
$hash = \%ENV;
$glob = \*STDOUT;
```

To create a reference to a subroutine:

```
sub foo { print "foo" };
$foosub = \&foo;
```

Anonymous Arrays

When you create a reference to an array directly - that is, without creating an intervening named array - you are creating an anonymous array.

Creating an anonymous array is easy:

```
$array = [ 'Bill', 'Ben, 'Mary' ];
```

This line assigns an array, indicated by the enclosing square brackets instead of the normal parentheses, to the scalar $array. The values on the right side of the assignment make up the array, and the left side contains the reference to this array.

You can create more complex structures by nesting arrays:

```
@arrayarray = ( 1, 2, [1, 2, 3]);
```

The @arrayarray now contains three elements; the third element is a reference to an anonymous array of three elements.

Anonymous Hashes

Anonymous hashes are similarly easy to create, except you use braces instead of square brackets:

```
$hash = { 'Man' => 'Bill',
          'Woman' => 'Mary,
          'Dog' => 'Ben'
};
```

Dereferencing

The most direct way of dereferencing a reference is to prepend the corresponding data type character ($ for scalars, @ for arrays, % for hashes, and & for subroutines) that you are expecting in front of the scalar variable containing the reference. For example, to dereference a scalar reference $foo, you would access the data as $$foo. Other examples are:

```
$array = \@ARGV;     # Create reference to array
$hash = \%ENV;       # Create reference to hash
$glob = \*STDOUT;    # Create reference to typeglob
$foosub = \&foo;     # Create reference to subroutine
push (@$array, "From humans");
$$array[0] = 'Hello'
$$hash{'Hello'} = 'World';
&$foosub;
print $glob "Hello World!\n";
```

Object Basics

There are three main terms, explained from the point of view of how Perl handles objects. The terms are object, class, and method.

- Within Perl, an object is merely a reference to a data type that knows what class it belongs to. The object is stored as a reference in a scalar variable. Because a scalar only contains

a reference to the object, the same scalar can hold different objects in different classes.

- A class within Perl is a package that contains the corresponding methods required to create and manipulate objects.
- A method within Perl is a subroutine, defined with the package. The first argument to the method is an object reference or a package name, depending on whether the method affects the current object or the class.

Perl provides a **bless()** function which is used to return a reference and which becomes an object.

Defining a Class

Its very simple to define a class. In Perl, a class is corresponds to a Package.To create a class in Perl, we first build a package. A package is a self-contained unit of user-defined variables and subroutines, which can be re-used over and over again. They provide a separate namespace within a Perl program that keeps subroutines and variables from conflicting with those in other packages.

To declare a class named Person in Perl we do:

```
package Person;
```

The scope of the package definition extends to the end of the file, or until another package keyword is encountered.

Creating and Using Objects

To create an instance of a class (an object) we need an object constructor. This constructor is a method defined within the

package. Most programmers choose to name this object constructor method new, but in Perl one can use any name.

One can use any kind of Perl variable as an object in Perl. Most Perl programmers choose either references to arrays or hashes.

Let's create our constructor for our Person class using a Perl hash reference;

When creating an object, you need to supply a constructor. This is a subroutine within a package that returns an object reference. The object reference is created by blessing a reference to the package's class. For example:

```
package Person;
sub new
{
    my $class = shift;
    my $self = {
        _firstName => shift,
        _lastName  => shift,
        _ssn       => shift,
    };
    # Print all the values just for clarification.
    print "First Name is $self->{_firstName}\n";
    print "Last Name is $self->{_lastName}\n";
    print "SSN is $self->{_ssn}\n";
    bless $self, $class;
    return $self;
}
```

Every method of a class passes first argument as class name. So in the above example class name would be "Person". You can try this out by printing value of $class. Next rest of the arguments will be rest of the arguments passed to the method.

Now Let us see how to create an Object

```
$object = new Person( "Mohammad", "Saleem", 23234345);
```

You can use simple hash in your consturctor if you don't want to assign any value to any class variable. For example

```
package Person;
sub new
{
    my $class = shift;
    my $self = {};
    bless $self, $class;
    return $self;
}
```

Defining Methods

Other object-oriented languages have the concept of security of data to prevent a programmer from changing an object data directly and so provide accessor methods to modify object data. Perl does not have private variables but we can still use the concept of helper functions methods and ask programmers to not mess with our object innards.

Lets define a helper method to get person first name:

```
sub getFirstName {
    return $self->{_firstName};
}
```

Another helper function to set person first name:

```
sub setFirstName {
    my ( $self, $firstName ) = @_;
    $self->{_firstName} = $firstName if defined($firstName);
    return $self->{_firstName};
}
```

Lets have a look into complete example: Keep Person package and helper functions into Person.pm file

```
#!/usr/bin/perl

package Person;

sub new
{
    my $class = shift;
    my $self = {
        _firstName => shift,
        _lastName  => shift,
        _ssn       => shift,
    };
    # Print all the values just for clarification.
    print "First Name is $self->{_firstName}\n";
    print "Last Name is $self->{_lastName}\n";
    print "SSN is $self->{_ssn}\n";
    bless $self, $class;
    return $self;
}
sub setFirstName {
    my ( $self, $firstName ) = @_;
    $self->{_firstName} = $firstName if defined($firstName);
    return $self->{_firstName};
}

sub getFirstName {
    my( $self ) = @_;
    return $self->{_firstName};
}
1;
```

Now create Person object in mail.pl fileas follows

```
#!/usr/bin/perl

use Person;

$object = new Person( "Mohammad", "Saleem", 23234345);
# Get first name which is set using constructor.
$firstName = $object->getFirstName();

print "Before Setting First Name is : $firstName\n";
```

```
# Now Set first name using helper function.
$object->setFirstName( "Mohd." );

# Now get first name set by helper function.
$firstName = $object->getFirstName();
print "Before Setting First Name is : $firstName\n";

This will produce following result
First Name is Mohammad
Last Name is Saleem
SSN is 23234345
Before Setting First Name is : Mohammad
Before Setting First Name is : Mohd.
```

Inheritance

Object-oriented programming sometimes involves inheritance. Inheritance simply means allowing one class called the Child to inherit methods and attributes from another, called the Parent, so you don't have to write the same code again and again. For example, we can have a class Employee which inherits from Person. This is referred to as an "isa" relationship because an employee is a person. Perl has a special variable, @ISA, to help with this. @ISA governs (method) inheritance.

Following are noteable points while using inheritance

- Perl searches the class of the specified object for the specified object.
- Perl searches the classes defined in the object class's @ISA array.
- If no method is found in steps 1 or 2, then Perl uses an AUTOLOAD subroutine, if one is found in the @ISA tree.
- If a matching method still cannot be found, then Perl searches for the method within the UNIVERSAL class (package) that comes as part of the standard Perl library.

- If the method still hasn't been found, then Perl gives up and raises a runtime exception.

So to create a new Employee class that will inherit methods and attributes from our Person class, we simply code: Keep this code into Employee.pm

```
#!/usr/bin/perl

package Employee;
use Person;
use strict;
our @ISA = qw(Person);    # inherits from Person
```

Now Employee Class has all the methods and attributes inherited from Person class and you can use it as follows: Use main.pl file to test it

```
#!/usr/bin/perl

use Employee;

$object = new Employee( "Mohammad", "Saleem", 23234345);
# Get first name which is set using constructor.
$firstName = $object->getFirstName();

print "Before Setting First Name is : $firstName\n";

# Now Set first name using helper function.
$object->setFirstName( "Mohd." );

# Now get first name set by helper function.
$firstName = $object->getFirstName();
print "After Setting First Name is : $firstName\n";

This will produce following result
First Name is Mohammad
Last Name is Saleem
SSN is 23234345
Before Setting First Name is : Mohammad
Before Setting First Name is : Mohd.
```

Method Overriding

The child class Employee inherits all the methods from parent class Person. But if you would like to override those methods in your child class then you can do it by givig your implementation. You can add your additional functions in child class. It can done as follows: modify Employee.pm file

```
#!/usr/bin/perl

package Employee;
use Person;
use strict;
our @ISA = qw(Person);    # inherits from Person

# Override constructor
sub new {
    my ($class) = @_;

    # Call the constructor of the parent class, Person.
    my $self = $class->SUPER::new( $_[1], $_[2], $_[3] );
    # Add few more attributes
    $self->{_id}   = undef;
    $self->{_title} = undef;
    bless $self, $class;
    return $self;
}

# Override helper function
sub getFirstName {
    my( $self ) = @_;
    # This is child class function.
    print "This is child class helper function\n";
    return $self->{_firstName};
}

# Add more methods
sub setLastName{
    my ( $self, $lastName ) = @_;
    $self->{_lastName} = $lastName if defined($lastName);
```

```
    return $self->{_lastName};
}

sub getLastName {
    my( $self ) = @_;
    return $self->{_lastName};
}

1;
```

Now put following code into main.pl and execute it.

```
#!/usr/bin/perl

use Employee;

$object = new Employee( "Mohammad", "Saleem", 23234345);
# Get first name which is set using constructor.
$firstName = $object->getFirstName();

print "Before Setting First Name is : $firstName\n";

# Now Set first name using helper function.
$object->setFirstName( "Mohd." );

# Now get first name set by helper function.
$firstName = $object->getFirstName();
print "After Setting First Name is : $firstName\n";


This will produce following result
First Name is Mohammad
Last Name is Saleem
SSN is 23234345
This is child class helper function
Before Setting First Name is : Mohammad
This is child class helper function
After Setting First Name is : Mohd.
```

Default Autoloading

Perl offers a feature which you would not find any many other programming languages: a default subroutine.

If you define a function called AUTOLOAD() then any calls to undefined subroutines will call AUTOLOAD() function. The name of the missing subroutine is accessible within this subroutine as $AUTOLOAD. This function is very useful for error handling purpose. Here is an example to implement AUTOLOAD, you can implement this function in your way.

```
sub AUTOLOAD
{
   my $self = shift;
   my $type = ref ($self) || croak "$self is not an object";
   my $field = $AUTOLOAD;
   $field =~ s/.*://;
   unless (exists $self->{$field})
   {
      croak "$field does not exist in object/class $type";
   }
   if (@_)
   {
      return $self->($name) = shift;
   }
   else
   {
      return $self->($name);
   }
}
```

Destructors and Garbage Collection

If you have programmed using objects before, then you will be aware of the need to create a .destructor. to free the memory allocated to the object when you have finished using it. Perl does this automatically for you as soon as the object goes out of scope.

In case you want to implement your destructore which should take care of closing files or doing some extra processing then you need to define a special method called **DESTROY**. This method will be called on the object just before Perl frees the memory allocated to it. In all other respects, the DESTROY method is just like any other, and you can do anything you like with the object in order to close it properly.

A destructor method is simply a member function (subroutine) named DESTROY which will be automatically called

- When the object reference's variable goes out of scope.
- When the object reference's variable is undef-ed
- When the script terminates
- When the perl interpreter terminates

For Example:

```
package MyClass;
...
sub DESTROY
{
    print "   MyClass::DESTROY called\n";
}
```

Another OOP Example

Here is another nice example which will help you to understand Object Oriented Concepts of Perl. Put this source code into any file and execute it.

```
#!/usr/bin/perl

# Following is the implementation of simple Class.
package MyClass;

sub new
{
   print "   MyClass::new called\n";
```

```
   my $type = shift;            # The package/type name
   my $self = {};               # Reference to empty hash
   return bless $self, $type;
}

sub DESTROY
{
   print "   MyClass::DESTROY called\n";
}

sub MyMethod
{
   print "   MyClass::MyMethod called!\n";
}


# Following is the implemnetation of Inheritance.
package MySubClass;

@ISA = qw( MyClass );

sub new
{
   print "   MySubClass::new called\n";
   my $type = shift;            # The package/type name
   my $self = MyClass->new;     # Reference to empty hash
   return bless $self, $type;
}

sub DESTROY
{
   print "   MySubClass::DESTROY called\n";
}

sub MyMethod
{
   my $self = shift;
   $self->SUPER::MyMethod();
   print "   MySubClass::MyMethod called!\n";
}

# Here is the main program using above classes.
package main;
```

```
print "Invoke MyClass method\n";

$myObject = MyClass->new();
$myObject->MyMethod();

print "Invoke MySubClass method\n";

$myObject2 = MySubClass->new();
$myObject2->MyMethod();

print "Create a scoped object\n";
{
        my $myObject2 = MyClass->new();
}
# Destructor is called automatically here

print "Create and undef an object\n";
$myObject3 = MyClass->new();
undef $myObject3;

print "Fall off the end of the script...\n";
# Remaining destructors are called automatically here
```

Previous Page | Next Page