OBJECT ORIENTED PROGRAMMING IN PERL

We have already studied references in Perl and Perl anonymous arrays and hashes. Object Oriented concept in Perl is very much based on references and anonymous array and hashes. Let's start learning basic concepts of Object Oriented Perl.

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.

- An **object** within Perl 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 which ultimately becomes an object.

Defining a Class

It is very simple to define a class in Perl. A class is corresponding to a Perl Package in its simplest form. 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.*

Perl Packages provide a separate namespace within a Perl program which keeps subroutines and variables independent 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 *anobject* 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 you can use any name.

You 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, which 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;
}
```

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 they provide accessor methods to modify object data. Perl does not have private variables but we can still use the concept of helper methods to manipulate object data.

Lets define a helper method to get person's first name −

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

Another helper function to set person's first name −

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

Now 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 let's make use of Person object in employee.pl file as 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";
```

When we execute above program, it produces the 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 has very good and useful concept called inheritance. Inheritance simply means that properties and methods of a parent class will be available to the child classes. So you don't have to write the same code again and again, you can just inherit a parent class.

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 the important points to be considered while using inheritance −

- Perl searches the class of the specified object for the given method or attribute, i.e., variable.
- 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 has not 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 as follows: 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 them 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";
```

When we execute above program, it produces the 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 the parent class Person. But if you would like to override those methods in your child class then you can do it by giving your own implementation. You can add your additional functions in child class or you can add or modify the functionality of an existing methods in its parent class. It can be 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 let's again try to use Employee object in our main.pl file 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";
```

When we execute above program, it produces the 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 in any other programming languages: a default subroutine. Which means, if you define a function called **AUTOLOAD,** then any calls to undefined subroutines will call AUTOLOAD function automatically. The name of the missing subroutine is accessible within this subroutine as $AUTOLOAD.

Default autoloading functionality is very useful for error handling. Here is an example to implement AUTOLOAD, you can implement this function in your own 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 object oriented programming 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 destructor, 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 method, and you can implement whatever logic you want inside this method.

A destructor method is simply a member function *subroutine* named DESTROY, which will be called automatically in following cases −

- 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, you can simply put the following method DESTROY in your class −

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

Object Oriented Perl Example

Here is another nice example, which will help you to understand Object Oriented Concepts of Perl. Put this source code into any perl 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
```

When we execute above program, it produces the following result –

```
Invoke MyClass method
MyClass::new called
MyClass::MyMethod called!
Invoke MySubClass method
```

```
MySubClass::new called
MyClass::new called
MyClass::MyMethod called!
   MySubClass::MyMethod called!
Create a scoped object
MyClass::new called
MyClass::DESTROY called
Create and undef an object
MyClass::new called
MyClass::DESTROY called
Fall off the end of the script...
MyClass::DESTROY called
MySubClass::DESTROY called
```