Stringification in classic Perl OOP

overload | ref | blessed | Scalar::Util

In the previous article we saw how to create a class using bless, but when we printed the instance object we got `My::Date=HASH(0x7f807c13c700)`. This can be useful as it tells us we are dealing with a My::Date object, but on the other hand it could print something more interesting. For example it could print a nice representation of the attributes: `Date(2013, 1, 27)`

Stringification

means that we take an object and create a string format. It happens when the object is placed in string context. For example when it is printed.

Operator overloading

In order to change the behavior of the My::Date class when an object of that type is placed in string context, we need to "overload" the stringification operator. It can be done using the overload module.

This is what needs to be added to the Date.pm file in the previous example.

```
use overload
    '""' => 'stringify';

sub stringify {
    my ($self) = @_;
    return sprintf 'Date(%s, %s, %s)', $self->year, $self->month, $self->day;
}
```

The `stringify` subroutine, that can actually have any name, implements the actual stringification. In some modules the author call it `to_string` or `to_str`, the actual name only matters because the method can also be used on its own. A user could write:

```
my $d = My::Date->new(year => 2013, month => 1, day => 27);
say $d->stringify;
```

This will print `Date(2013, 1, 27)`.

The first two lines in the above code, loads the overload module and tells it that when the object is in string context, call the stringify method. When "use"-ing the overload module we pass a set of key-value pairs. In this case the key is `""` (two double-quotes) and the value is the name of the method that implements it.

The more interesting usage of this is when we just simply print `$d`:

```
my $d = My::Date->new(year => 2013, month => 1, day => 27);
say $d;
```

In this case too, perl will call the `stringify` method of the `My::Date` class, and will print the value returned by that function.

Which name-space does the object belong to?

Now that we change what a single object returns, we seem to have lost the previous information. To which class does this object belong to?

No problem, the `ref` function can tell us this information:

```
say ref $d;
```

will print `My::Date`.

Alternatively, one can use the `blessed` function of Scalar::Util

```
use Scalar::Util qw(blessed);
say blessed $d;
```

The advantage of the `blessed` function is that it will return undef if the values is not blessed, while `ref` can return values such as `HASH`, `ARRAY` etc. So if there is a scalar variable that you want to know if it is blessed or not, it is simpler to write

```
if (defined blessed $var) {
}
```

that to compare the result of `ref` to all the known types of references.

Full example

```
package My::Date;
use strict;
use warnings;

sub new {
```

```
        my ($class, %args) = @_;
        return bless \%args, $class;
    }

sub year {
        my ($self, $value) = @_;
        if (@_ == 2) {
            $self->{year} = $value;
```