Objected Oriented Perl

An introduction – because I don't have the time or patience for an in-depth OOP lecture series…

Before we can begin…

- Let's talk about references
- We've used them a few times before
  - 2D arrays, parameters to subroutines
- Never made anything formal
- Now we will, because without them, Perl objects can't exist

References

- A reference is a pointer to a 'real' variable.
- We've seen how to create references to existing variables:
  - $array_ref = \@array;
  - $hash_ref = \%hash;
  - $scalar_ref = \$scalar
- Can also create Anonymous references…

Anonymous References

- These are pointers to values stored in memory that aren't yet (or ever) labeled by a variable name.
- Syntax is similar (but different) to creating actual variable
- Array:
  - $array_ref = [1, 2, 3, 4, 5];
- Hash:
  - $hash_ref = {Name => 'Paul', ID => 123};
- Scalar:
  - $str_ref = \"A String\n";
  - $int_ref = \456;

Dereferencing References

- We've seen before how to dereference entire arrays or hashes (and scalars):
  - @array = @$array_ref;
  - %hash = %$hash_ref;
  - $scalar = $$scalar_ref;
- Depending on your point of view, dereferencing members of array- or hash-references is either easier or more complicated…

TMTOWTDI

- In fact, there are three…
- `$ $a_ref  [2] = "Hello";`
- `${$a_ref} [2] = "Hello";`
- `  $a_ref->[2] = "Hello";`

- `$ $h_ref  {$key} = $value;`
- `${$h_ref} {$key} = $value;`
- `  $h_ref->{$key} = $value;`
- These are all valid and acceptable. Form you choose is whatever looks the best to you.

Okay, Let's Get Started

- This will be an introduction only. We will cover the very basics. Anything more complicated goes beyond the scope of this course
  - In fact, even this introduction is treading pretty close to the edge of that scope.

Classes in Perl

- A class is defined by storing code which defines it in a separate file, and then `use`ing that file
- The file must be named with the name of the class (starting with an capital letter), followed by the extension `.pm`
- After the shebang in your 'main' file, this line of code:
- `use <Classname>;`
- You can then create instances of the class anywhere in your file.

Defining an Object

- In Perl, an object is simply a reference containing the members of a class.
  - typically, a reference to a hash, but can be any kind of reference
- The reference becomes an object when it is "blessed" – when you tell Perl the reference belongs to a certain class.

Simple Example

```
package Student;
$obj = {Name => 'Paul',
        ID => 123};
bless ($obj, Student);
```

- $obj is now an object of the class Student
- 'package Student' is the first line of your .pm file. It identifies all following code as belonging to this class/package/module

Constructors

- Unlike C++, a Constructor in Perl is simply another subroutine. Typically named 'new', but you can give it any name you want.

```
package Student;
sub new {
  my $ref = {Name => "", ID => 0};
  bless ($ref, Student);
  return $ref;
}
```

- In this example, don't actually have to give $ref any elements. You can define them all in a later subroutine, if you choose.

Calling the Constructor

- As you may be able to guess, TMTOWTDI
- `$student = new Student;`
- `$student = Student->new;`
- `$student = Student::new(Student);`
- First two methods get translated to 3rd method internally by perl. This has beneficial consequences…

Arguments to Constructor

- (actually, this applies to arguments to any method)
- Every time the constructor is called, first argument to function is the name of the class.
- Remaining arguments are caller-defined
- `$obj = new Student (“Paul”, 123);`
- `$obj = Student->new(“Paul”, 123);`
- `$obj = Student::new(Student, “Paul”, 123);`
- So, when defining constructor, often see this:

```
sub new{
  my $class = shift;
  my ($name, $ID) = @_;
  my $ref = {Name => $name,ID => $ID};
  bless ($ref, $class);
  return $ref;
}
```

More Methods

- Within the .pm file, any subroutines you declare become methods of that class.
- For all methods, first argument is always the object method is being called on. This is also beneficial…

```
sub setName{
  my $ref = shift;
  my $name = shift;
  $ref->{Name} = $name;
}
```

- To call this method:
- `$obj->setName(“Paul Lalli”);`
- Perl translates this to:
- `Student::setName($obj, “Paul Lalli”);`

One more thing…

- In one of the oddest things I’ve learned about Perl, you need to place the following statement at the end of your .pm file:
- `1;`
- This is because the ‘use’ keyword needs to take something that returns a true value. Perl ‘returns’ the last statement evaluated.

♫♪ Be Kind… to One Another♪♫

- Note that class variables are not strictly ‘private’ in the C++ sense.
- There is nothing preventing the user of your class from modifying the data members directly, bypassing your interface functions.
- Perl’s general philosophy is “If someone wants to shoot himself in the foot, who are you to stop him?”
- When using other people’s classes, almost always a better idea to use the functions they’ve given you, and pretend you can’t get at the internal data.
- There are, of course, methods you can use to prevent users from doing this.
  - Significantly beyond scope of this course
  - Really not worth the trouble