Object Oriented Programming

Programming Perl
Chapter 12: "Objects"

Tuesday's Homework (Processing addresses)

Two style suggestions

Store mixed data in a hash

```
my $record = [$addr,$state,$zip];
$record[1] # Less clear

my $record = {
  'address' => $addr,
  'state' => $state,
  'zip' => $zip,
$record{'zip'} # More clear
```


Memory is cheap

- Make multiple copies of data if it's easier to work with

```
my %name_index;
my %state_city_index;
```

- Counterpoint: multiple copies makes it easier to diverge

Store parsed data, not raw strings

```
$name_index{$name} = $line;
```

– Harder to use later

```
$name_index{$name} = {
  'state' => $state,
  'city' => $city,
}
```

– Easier to use later

Wednesday's Homework

Interesting problems

Object Oriented Programming

A simple module:

```
use FindBin;
print "I was run from
  $FindBin::Bin\n";
```

- There is a module called "FindBin", and it contains a variable $Bin.

A quick trick

- My module is next to my script
  - (ex path/main.pl)

```
% ./main.pl
Hello, world!
% cd ..
% path/main.pl
Can't locate Alan/Hello.pm in
  @INC
```


Why?

- My directory isn't in the search path
- Can set the environment variable PERL5LIB
- Easier:

```
use FindBin;
use lib $FindBin::Bin;
```

- "`use lib 'path'`" lets you add to Perl's search path
- FindBin returns the the directory your script is in

Another module:

```
use Digest::MD5;
my $encoded =
  Digest::MD5::md5_hex($password);
```

- Digest::MD5 provides the function md5_hex

Packages

- Obviously Perl provides some sort of namespaces (C++) or packages (Java).

package

`package Alan::MyPackageName;`

- Everything from that point forward is in the Alan::MyPackagename package.
- Stop at end of file, or the next package line.
- You start in a package cleverly called main.

package

- A single file can hold multiple packages
  - (ex. two-in-one.pl)

```
A::hi();
B::hi();
package A;
sub hi { print "Hello from A\n"; }
package B;
sub hi { print "Hello from B\n"; }
```


package per file/module

- Put a package in a file with of the same name:
  - (ex hello_world.pl, Alan/Hello.pm)
- In the file "Alan/Hello.pm":

```
package Alan::Hello;
sub hi { print "Hello, world!"; }
1;
```

- In "hello_world.pl":

```
use Alan::Hello;
Alan::Hello::hi();
```


That's all nice, but where at the objects?

A module implementing an object

```
use FileHandle;
my $fh = new FileHandle;
$fh->open("</etc/services");
my $line = $fh->getline;
$fh->close;
```

Perl's Do-It-Yourself Object System

- package
- module
- blessing

bless my reference

- An object is a reference that is "blessed" into a package
- Usually the package is in a file/module of the same name

An example

(ex. oo/before and oo/after)

Common (simplified) usage

```
sub new {
  my($class, $name) = @_;
  my($self) = {
    'name' => $name,
    'count' => 0,
  }, $class;
  bless $self, $class;
  return $self;
}
```


Common (simplified) usage

```
sub new {
  my($class, $name) = @_;
  return bless {
    'name' => $name,
    'count' => 0,
  }, $class;
}
```


Create your own idiom?

- "new" isn't special
- You could call it "create" or "fred".
- new is a good name. Use that.

Alternate calls

- `alan->new();`
- `new alan;`
- `$existing_instance->new();`

$existing_instance->new();

- Requires a tweak to support...

```
sub new {
  my($class, $name) = @_;
  return bless {
    'name' => $name,
    'count' => 0,
  }, (ref($class) || $class);
}
```


Python

```
class hello(object):
  def __init__(self, name):
    self._count = 0
    self._name = name

  def hi(self):
    self._count += 1
    print "Hello, %s (I've said hello to %s
  %d times)" % \
      (self._name, self._name, self._count)

hello_alan = hello("Alan")
hello_alan.hi()
```


Ruby

```
class Hello
  def initialize(name)
    @name = name
    @count = 0
  end

  def hi
    @count = @count + 1
    puts "Hello, #{@name} (I've said hello to
  #{@name} #{@count} times)"
  end
end

hello_alan = Hello.new("Alan")
hello_alan.hi
```


Javascript

- Function as class

```
function hello(name) {
  this.name = name;
  this.count = 0;

  this.hi = function() {
    this.count++;
    document.write("Hello, " + this.name + " (I've
  said hello to " +
      this.name + " " + this.count + " times)\n");
  }

}

hello_alan = new hello("Alan");
hello_alan.hi();
```


Advanced Topics

Inheritance

```
package HondaCivic;
@ISA = ("Car");
```

- A HondaCivic "is a" Car.
- If HondaCivic fails to implement a method, try car
  - (ex. inherit/*)

Freeing object memory

- General rule: don't worry about it
- More specific rule: ensure no variables point to an unwanted object, it will go away eventually
  - my $obj = new MyObject;
  - $obj = undef;
- Most specific rule: Break circular references
  - Many scripting languages only reference count and don't fully garbage collect