CS 3101-4 - Programming Languages: Perl

Lecture 6: More on Objects, CPAN, Review

Vinodkumar Prabhakaran
vinod@cs.columbia.edu

April 26, 2012

Contents

Announcements

General

Course evaluation

- available on courseworks.
- deadline is Monday April 30th.

Availability next week

- no office hours on Monday April 30th.
- will hold office hours tomorrow, Friday April 27, 2pm-4pm.
- available via email for any questions

Submissions

- Homework 3
  - questions?
- Homework 4
  - submit by Friday 11.59pm - no extensions
  - grades will be posted over the weekend
- Project
  - submit via courseworks as one tar.gz file (just like homeworks)
  - by Friday, May 4th, 11.59 pm EST

Project submission

- Submission should include
  - project report
  - entire code base with documentation
- Grading
  - Functionality (expectation will be higher for 2-person teams) as per the proposal: 15%
  - Documentation (in code documentation) and README: 5%
  - Report: 10%
  - Efficiency, readability, good programming practices: 10%

Project submission

- Report
  - 4-5 page report (1-person team)
  - 7-9 page report (2-person team)
  - what the project is about
  - should describe functionality in detail
  - relate it back to the proposal - how much was done, what was skipped, what was added etc.
  - should list third party or CPAN modules used.

Review

- Packages
  - partition namespaces
- Modules
  - packages in separate files, with .pm extensions
  - Exporter module
  - use vs. require
- Classes and Objects
  - creating objects and blessing
  - constructors and destructors
  - methods - class vs. object methods
  - inheritance - single and multiple

Recap

- Object
  - a reference, which could be to a hash, array or even a scalar.
  - it holds the 'instance data'
- Class
  - a package with a set of subroutines to execute class's methods
  - package variables hold 'class data'
  - a module may contain one or more classes (packages)
- Method
  - subroutines in the package that is used as the class.
  - method invocation (another way of calling these subroutines) passes an extra argument: the object or package name
    - $mycar->driveto("vermont")
    - Car::driveto($mycar,"vermont")

Method Invocation

- Arrow operator: invocant->method(arguments)

```
# what we saw in lec5
$bankaccount = BankAccount->new();
$bankaccount->deposit(100);
$bal = $bankaccount->balance();
# left associative
BankAccount->new()->deposit(100);
```

- Indirect invocation: <method invocant arguments>

```
$bankaccount  =  new BankAccount;
deposit $bankaccount 100;
$bal = balance $bankaccount;
```

fields pragma

- enables compile-time verified class fields.
- specifies the list of fields in the object
- fields::new() creates and blesses a pseudo-hash comprised of the fields declared using the fields pragma into the specified class.

```
package Person;
use fields qw(name age address);
sub new {
  my $name = shift;
  $self = fields::new($name);
  $self->{name} = "";
  $self->{age} = 0;
  return $self;
}
```

fields pragma

```
package Person;
use fields qw(name age address);
sub new {
  my $name = shift;
  $self = fields::new($name);
  $self->{name} = "";
  $self->{age} = 0;
  return $self;
}
```

in effect does

```
package Person;
sub new {
  my $name= shift;
  $self = {name => "", age => 0, address => ""};
  bless ($self, $name);
  return $self;
}
```

fields pragma

- helps prevent mis-spelt field names.

```
package Person;
use fields qw(name age address);
sub new {
  my $self = shift;
  $self = fields::new($self) unless ref $self;
  $self->{name} = "";
  $self->{age} = 0;
  return $self;
}
sub some_method {
  $self = shift;
  $self->{naem} = "Alex";
  # compiler throws an error here. But, in the
      normal setup this is not an error; Perl
      would just add a new key named "naem" into
      the $self hash.
}
```

use base pragma

- if an object is implemented using the 'use fields' pragma, all participants in the inheritance hierarchy must use 'use base' and 'use fields' declarations

```
package Student;
use base "Person";
use fields qw(grade year classes);
```

- makes Student a subclass of Person
- loads Person.pm file
- adds three new variables in Student class along with those from Person class

```
$self = fields::new("Student");
$self->{name} = ""; # inherited from Person
$self->{year} = 1; # of Student
```

- NOTE: Inheritance using '@ISA' inherits only methods.

Class::Struct

- a standard module that helps to start off with a new class
- exports a function 'struct'

```
package Person;
use Class::Struct;
struct Person => {
  name => '$', # name is a scalar
  age => '$', # age is a scalar
  address => '%', # address is a hash ref
};
1;
```

- generates a constructor named 'new'
- generates accessor methods for each of the instance data fields named in the structure

Class::Struct

- generates a constructor named 'new'
- generates accessor methods for each of the instance data fields named in the structure

```
$person = Person->new();
$person->name("Alex");
$person->age(29);
$person->address({line1 => "475 Riverside", City
    => "New York", Zip => "10115"});
```

Class data

- to store a common state shared by all objects of a class.
- global to entire class
- e.g. count of all objects, location of log file etc.
- static variable (in C++, Java etc.)
- no separate syntax in Perl, but various options
  - store as a package variable, with varying levels of 'protection'
  - store the references in the object

CPAN

- Comprehensive Perl Archive Network
- central repository of hundreds of Perl modules
- ▸ Link

Installing a CPAN module

- download the tar.gz from CPAN website
- unzip and go into that directory
- build

```
perl MakeFile.pl
make
make test
```

- install

```
make install
```

An easier alternative

- install cpanm

```
cpan App::cpanminus
```

- install any module using

```
cpanm Module::Name
```

Review

- Basic data types: scalars($), arrays(@), hashes(%)
- scalar vs list context
- quotes and interpolation
- operators: numeric vs string
- control: if-elsif-else, unless, while, until, for, foreach
- statement modifiers (print $line if $valid;)

Review

- Input/Output, the magic while(<>) { ... }
- array operations: push, pop, shift, unshift, delete
- array operations: split, join, sort, reverse
- hash operations: keys, values, each, delete, reverse
- hash operations: exists, defined
- Subroutines: define, return, args (@_)

Review

- scopes: `my` vs `local`
- pragmas: warnings, strict, fields etc.
- Pattern matching
  - matching and substituting
  - greedy vs. non-greedy matching
  - http://files.meetup.com/120908/Regexp Quick Reference.pdf

    ▸ Link

Review

- References
  - references to scalars, arrays and hashes
  - anonymous references [ ], { }
  - The arrow operator -> to access references
- Data structures
- Subroutine references
- Symbolic references

Review

- Packages: partitioned namespaces
- Modules: use vs. require
- Classes and Objects
  - blessing, constructors
  - data - class vs. object data
  - methods - class vs. object methods
  - inheritance - single and multiple
  - fields and base pragma
  - Class::Struct
- using CPAN modules

Common mistakes

The file handle in print

```
print OUT, a, b, c;   # throws an error

open ($fh, ">ttt");
print $fh "YYYYY", "\t", "ZZZZ";   # works fine

print $fh, "YYYYY", "\t", "ZZZZ"; # wrong, no error
# prints to STDOUT
# GLOB(0x100804ed0)YYYYY ZZZZ
```

The == vs eq

```
$a = "123";
$b = "123.00";

if ($a eq $b) # false

if ($a == $b) # true
```

Common mistakes

Blocks require braces

```
if ($a > $b)
  print "yay";
#  valid for C, Java etc. not for Perl.

if ($a > $b) {
  print "yay";
}
```

Using the proper preamble for variables

```
@arr[1]; # an array slice with just one element
$arr[1]; # an array element

print "Element @arr[1]\n"; # works the same

@arr[1] = <INFILE>;
# enforces list context
# assign first line to $arr[1]; discard rest
```

Reminders

Keep in mind the context

```
($x) = (aa,bb,cc); # $x gets the value "aa";

$x = (aa,bb,cc); # $x gets the value "cc";

@arr = (aa,bb,cc);
$x = @arr; # $x gets the value 3;
```

The magic while

```
# <FH> auto assigns to $_ only within a while, that
    too when used as the only thing in the while
    condition

while (<FH>) { } # $_ gets each line

<FH>; # one line is read and discarded
```

Reminders

- => is the same as ,

```
%hash = (Jim => 45, Kim => 42);
# same as
%hash = (Jim, 45, Kim, 42);
```

- array/hash vs reference to array/hash

```
@array = (45, 42); # creates an array
$arrayref = [45, 42]; # creates an array reference
$array[0];
$arrayref->[0];

%hash = (Jim => 45, Kim => 42); # creates a hash
$hashref = {Jim => 45, Kim => 42}; # creates a
    hash reference
$hash{Jim};
$hashref->{Jim};
```

Regex tips

- save your regex variables; every new m// or s/// will reset the $1, $2, ... $‘, $&, $’ etc.

```
$_ =~ /(\w+)\s+(\w+)/;
if ($1 =~ /^X/) {
  print "yay";
}
print "$1 $2"; # prints nothing

my ($one, $two) = $_ =~ /(\w+)\s+(\w+)/;
```

- stay away from $‘, $& and $’ if possible
- if your program contains at least one instance of any of these, it causes all matches to keep these for future reference
- so, if once used, use as many times as you wish

Efficiency tips

- use hashes instead of linear searches

```
# search $word in @keywords

#simple method, but repeated for every $word
foreach (@keywords) {
  if ($word eq $_) { print "yes"; }
}

# more efficient, the loop is executed only once
%keywords;
for (@keywords) {
  $keywords{$_}++;
}

if ($keyword{$word}>0) { print "yes"; }
```

More tips

- my variables are faster than local variables
- print with comma separator is faster than '.' concatenation

```
print $person{$name}, " is ", $person{$age}, "
    years old";

# faster than

print $person{$name}." is ".$person{$age}." years
    old";
```

- always use the 'use warnings' pragma to save time on debugging
- use the 'use strict' pragma whenever possible

- Thank you
- Course evaluation (in courseworks)
- Best of luck with projects