Survey of Advanced Perl Topics

Database access
"OpSys"-like functions
Signal Handling
Inside-Out Objects

Database Access

- Standardized through the DBI module
  - not core module, but installed on CS system
- Differing Database systems have their own DB driver module
  - DBD::mysql, DBD::Oracle, DBD::Informix, etc
- By simply changing which DBD::* you use, your existing code can work with a new database.
- Setting up an individual DB on CSNet is difficult, due to lacking root access. If you really need one, contact labstaff.

General Format of Perl DBI

- declare a "DataBase Handler" to connect to the Database
- define SQL statement(s)
- prepare the SQL, returning "STatement Handler(s)"
- execute the statement handlers, passing values for the bind parameters
- fetch the results from the statement handlers, if apporpriate

DBI Example

```
use DBI;
my $dsn =
'DBI:mysql:database=game;host=db.example.com';
my $dbh = DBI->connect($dsn,$user,$password);
```

```
my $sql = "SELECT * FROM players ";
$sql   .= "WHERE score > ? AND score < ?";
```

```
my $sth = $dbh->prepare($sql);
$sth->execute($minscore, $maxscore);
```

```
while (my $r = $sth->fetchrow_hashref){
  print "$r->{name}'s score: $r->{score}\n";
}
```

- \- You can use the same $sth to now execute the prepared SQL with different values

Fetching Methods

- **`fetchrow_hashref`**
  - fetch "next" row, as a hash reference (key = column name, value = field value)
- **`fetchrow_arrayref`**
  - fetch next row, as an array reference (in order defined by the table)
- **`fetchrow_array`**
  - fetch next row as an array
- **`fetchall_arrayref`**
  - fetch entire results
    - no parameters - entire table as arrayref of arrayrefs
    - pass array ref - select numbers of which columns of entire table to return
    - pass hashref - select names of which columns to return, and return as an array of hash referneces
    - pass empty hashref - return entire table as array of hash references
- **`fetchall_hashref($key)`**
  - fetch entire results, as a hashref of hashrefs. Key is index of table

DBI errata

- **`do()`** - combines **`prepare()`** and **`execute()`**.
  - if you don't need to repeatedly execute the statement
  - Frequently used for DELETE and INSERT statements
    - can't be used for SELECT, because there's no $sth to fetch from
  - **`$dbh->do('DELETE FROM class WHERE drop = 1');`**
- **`$sth->rows()`** returns the number of rows inserted, updated, or deleted.
  - DOES NOT return number of rows that are selected!
- SQL NULL ==> Perl **`undef`**
- **`$dbh->{RaiseError} = 1`**, Perl will die on any SQL error.
  - (Otherwise, must check return value of every db call, and then check **`$DBI::err`**)
- http://dbi.perl.org & **`perldoc DBI`**

"OpSys-like" functions

- touched on these in the "external commands" presentation that we didn't cover in lecture.
  - basically said "Don't do that".
- please, take extreme caution when using these functions
- listing of Perl equivalents only
- for more information about the internals of Unix, take OpSys

fork()

- split off a separate process, duplicating the code and environment
- return value in parent is child's new pid
- return value in child is 0
-
```
my $pid = fork();
if ($pid) { #parent
  print "Child $pid just forked off\n";
  do_parent_stuff();
} else {
  print "I'm the child, just spawned\n";
  do_child_stuff();
}
```

`wait()`

- wait for one of the child processes to finish
- returns the PID of the child that just finished
- **`$?`** is set to the exit status of the child that was just found with wait
- **`waitpid($pid, 0)`**
- wait for a specific child to exit

exec($cmd, @args)

- Execute **`$cmd`**, passing **`@args`** to that command
- executes IN THE CURRENT PROCESS, wiping out anything else this code was going to do
- therefore, has no return value
- any code below the **`exec`** (other than a warning that the exec failed) is meaningless.
- **`$retval = system($cmd, @args);`** is equivalent to:
-
```
if (my $pid = fork()){ #duplicate process
    waitpid($pid);#parent wait for child to exit
    $retval = $?; #exit status of child
} else {
    exec ($cmd, @args); #child executes $cmd
}
```

Signal Sending

- Processes can send signals to one another.
- Most common use is to tell a process to die
- Therefore, function to do this is **`kill`**
- **`kill($signal, $pid);`**
- to see which signals are available, run **`` `kill -l` ``** on your system
- By default, most (all?) signals result in program termination
- Most shells respond to a CTRL-C by sending the current process a **`SIGINT`**

Signal Handling

- With the exception of SIGKILL (9) and SIGSTOP (23), all signals can be caught and processed.
- If your program receives a certain signal, you can decide what to do about it.
- Assign a reference to the handler subroutine to the %SIG hash, where the key is the 'name' of the signal
  - signal name also passed as first argument to the subroutine.
-
```
$SIG{INT} = sub {
  print "Bwaha, your CTRL-C doesn't scare me!";
};
```
- Now, if the user tries to CTRL-C your program, the message will be printed out instead of the program dieing.
  - (To actually kill this script, find out its pid from `ps -u <rcsid>`, and then send it a SIGKILL: `kill -9 <pid>`)

Inside-Out Objects

- One of the biggest problems with Perl objects is that there is no privacy.
  - users may directly access internals, ignoring class interface
- Also affords no protection from typos
  - **`$student->{naem}`** doesn't result in any compilation error.
- Most popular and well-developed solution to these problems is "inside-out objects"
  - completely reverse your way of thinking about how Perl defines a class and its objects.

Turn yourself around...

- In "normal" system, a class is created as a set of references to hashes.
  - One object == one reference to a hash
  - Each attribute is a key to each hash
  - Each attribute value is the value of that key in each hash
- An Inside-Out class is created as a set of references to anonymous scalars
  - One object == one reference to one scalar
  - Each attribute is one hash in the class
  - Each attribute value is the value of that object in that hash.

An example

```
package Student;
use Scalar::Util qw/refaddr/;
{
  my %name_of;
  my %rin_of;
  my %grade_of;
  sub new {
     my $class = shift;
     my ($name, $rin, $grade) = @_;
     my $ref = \do { my $scalar };
     $name_of{refaddr $ref} = $name;
     $rin_of{refaddr $ref} = $rin;
     $grade_of{refaddr $ref} = $grade;
     bless $ref, $class;
  }
  # other methods...
}
```

Object Methods

```
sub set_grade {
    my $obj = shift;
    my $grade = shift;
    $grade_of{refaddr $obj} = $grade;
}
sub status {
    my $obj = shift;
    if ($grade_of{refaddr $obj} >= 60) {
       return 'passing';
    } else {
       return 'failing';
    }
}
```

- Calls to these methods are exactly the same as they would be when using "normal" objects.

Benefits

- Object returned to the caller now has no link of any kind to the attribute values. User has no way to "peek" into internal structure, nor to directly modify internal structure
- Typos of attribute names are now either typos of actual variables (within the class) or method names (external to the class), and hence report compilation or runtime errors
- When using inheritance, both base class and children classes can use same attribute name without conflict.

Drawbacks

- When object (ie, scalar ref) goes out of scope, its attributes no longer are destroyed automatically
  - the hash containing the attributes is still in scope
  - Provide an explicit destructor for all I-O objects

```
sub DESTROY {
    my $obj = shift;
    delete $name_of{refaddr $obj};
    delete $rin_of{refaddr $obj};
    delete $grade_of{refaddr $obj};
}
```

- No built-in way to examine the entire internal state of the object for debugging purposes
  - which is, of course, what we were going for...

Class::Std

- CPAN module written by Damian Conway, author of *Perl Best Practices*
- Used with Inside Out Objects, to provide framework for all classes.
- Provide **`:ATTR`** trait for all attribute hashes
  - will automatically be cleaned up, no need for an explicit **`DESTROY`**
- provides global **`new()`** constructor, which automatically calls **`BUILD()`** subroutine for your class and all of its parents
  - **`BUILD()`** called with object, identifier, and argument hashref

Standard Class

```
• package Student
  use Class::Std;
  {
•     my %name_of  :ATTR;
      my %rin_of   :ATTR;
      my %grade_of :ATTR;
•     sub BUILD {
        my ($obj, $ident, $arg_ref) = @_;
•       $name_of{$ident}  = $arg_ref->{name};
        $rin_of{$ident}   = $arg_ref->{rin};
        $grade_of{$ident} = $arg_ref->{grade};
•       return;
      }
  }
• In main code:
  my $stu = Student->new(
    { name => 'Paul', rin => 1234, grade => 90 }
  );
```

More Class::Std

- Also exports an **`ident`** method, which does the same thing as **`Scalar::Util::refaddr`**
  - given an object, returns that object's numeric unique identifier
- Gives every class a **`_DUMP()`** method, so you can see what each object's attributes are currently set to.
  - Similar to function **`Data::Dumper`** provides for "normal" classes.
  - Shows values from each hash defined with the **`:ATTR`** trait

Automate even more

- The **`:ATTR`** trait can take several key=>value pairs to reduce mundane typing:
- **`my %rin_of :ATTR(get => 'rin');`**
  - automatically provides a standard get_rin() sub
- **`my %rin_of :ATTR(set => 'rin');`**
  - automatically provides a standard set_rin() sub
- **`my %rin_of :ATTR(init_arg => 'rin');`**
  - automatically **`BUILD()`**'s the rin attribute from the argument list
  - if **`BUILD()`** only initializes the attribute hashes, no need to write an explicit one
- You may combine any or all of these key/value pairs.
  - **`my %rin_of :ATTR(get=>'rin', set=>'rin');`**
    - use **`name => 'rin'`** as shortcut for all three of above
  - Due to a limitation (ie bug) in Perl, entire **`:ATTR`** string must be on one line of code.

Recommended Further Reading

- *Advanced Perl Programming*, Simon Cozens
  - Introspection, parsing beyond Regexps, Templating, Databases, Unicode, Event Driven Programming, Embedding C in Perl
- *Higher Order Perl*, Mark Jason Dominus
  - Recursion & Callbacks, Dispatch Tables, Iterator-based programs, Parsing & Infinite Streams, "Higher-Order" functions, Program transformations
- *Perl Best Practices*, Damian Conway
  - Tips and Advice for writing clean, elegant, maintainable Perl programs