

Advanced Bioperl

Some Module & Reference Help

- In any of the code, if you don't know what type of Object you are getting
- `print ref($obj)`
- `use Data::Dumper;`
`print Dumper($obj);`
- Try this for `Bio::SearchIO;`

Let's get to the Details

- What are Perl Modules about?
- How might I extend them?
- Details about Bioperl design and how to extend it.

Object Oriented Programming

- Break pieces of a problem down into components
- These objects can inherit from each other so they have behavior of parent, plus more
- Objects (class) will have defined set of entry points (methods) and associated data (state)

Perl object oriented structure

- Perl is not explicitly typed - so objects per-se don't exist.
- Perl OO is a type of hack, but it works!
- Perl6 will probably fix this, but don't hold your breath.

Perl Modules

- Modules are collections of sub routines, plus some associated data
- Two weird things to it:
 - ‘package’ defines name of package
 - ‘bless’ is used to bless an object as a package

Typical module

```
package MyPerson;
use strict;
use vars qw(@ISA); # for inheritance, can omit

sub new {
    my ($class, @args) = @_;
    my ($first,$last,$id,$father,$mother) = @args;
    my $self = bless {
        'fname' => $first,
        'lname' => $last,
        'id'     => $id,
        'mother' => $mother,
        'father' => $father,
    }, $class;

    return $self;
}
sub father {
    my ($self,$val) = @_;
    return $self->{'father'};
}
sub id { return shift->{'id'} }
```

Using that module

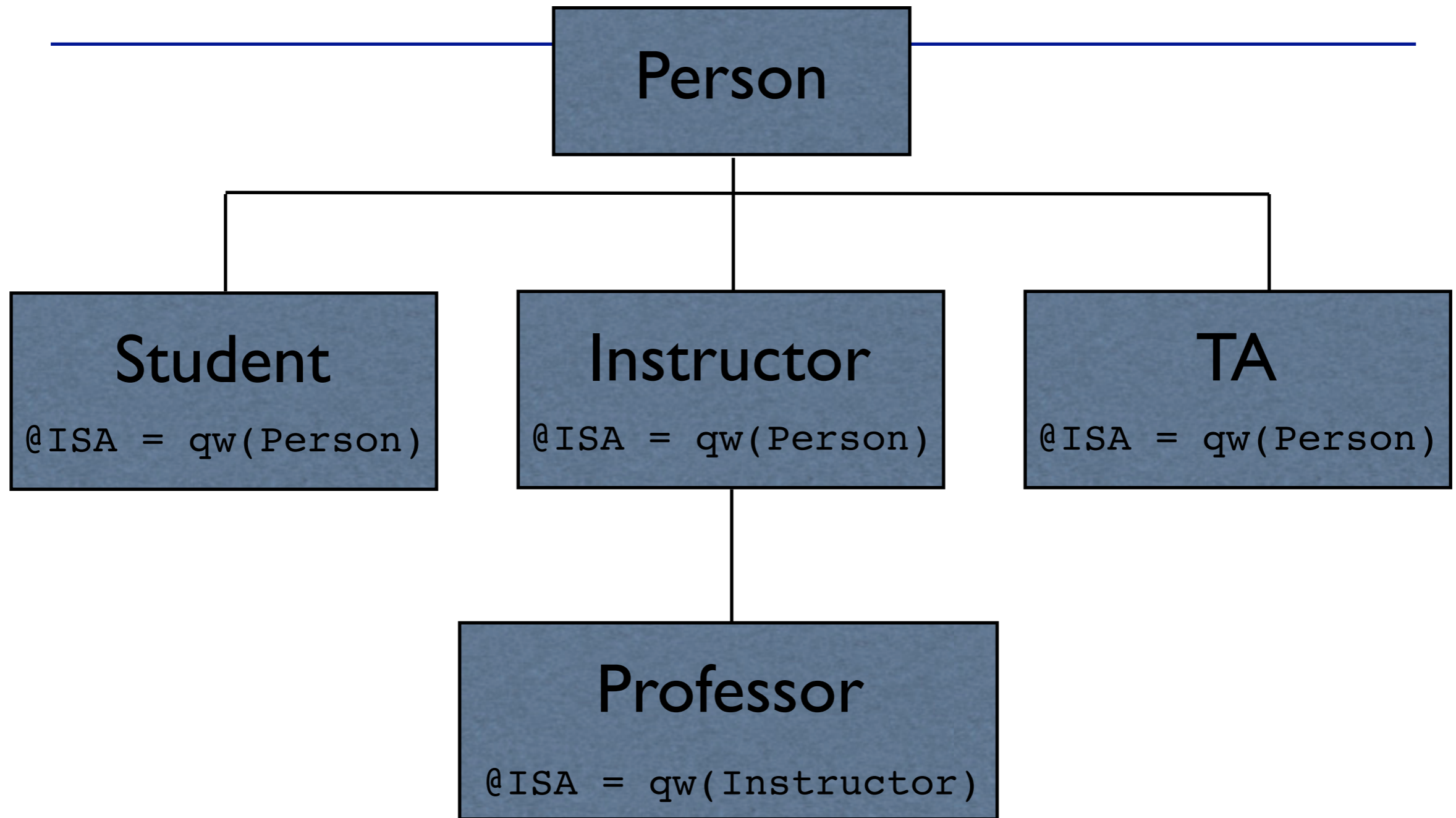
```
#!/usr/bin/perl -w
use strict;
use MyPerson;
my $person = MyPerson->new('jimbo', 'gumbo', 1,2,3);
my @family = (undef,$person); # 0 index needs to be empty
push @family, MyPerson->new('mom', 'gumbo', 3, 0, 0);
push @family, MyPerson->new('dad', 'gumbo', 2, 0, 0);
my $dad = $family[$person->father];
print "father id is ", $person->father, "\n";
print "father obj id is $dad id is ", $dad->id, "\n";
```

Perl OO tips

- Namespace (NAME::SUBNAME)
- Does not mean anything about inheritance, but good programmers will group things together
- @ISA or 'use base' is where inheritance is defined
- Use 'self' as reference point (like 'this' for C++/Java programmers)
- Thinks of modules as bags of methods

Extending a module

Inheritance



Practically

```
package Student;
use strict;
use base Person;

sub new {
    # respect the same arguments as your parent
    # or follow Bioperl convention
}
sub homework { }
sub bus_schedule { }
```

Bioperl Design

Bioperl Interfaces

- Describe the contract of what methods will be implemented
- Modules end in 'I'
- Kind of hacky - and sometimes ignored
- But can be useful and powerful - Ensembl, Bio::DB::GFF examples where basic interfaces are reused

Bioperl objects

- Seqs: Bio::PrimarySeq, Bio::Seq, Bio::Seq::RichSeq
- Features: Bio::SeqFeature, Bio::SeqFeature::Generic
- Annotations: Bio::Annotation::Collection, Bio::Annotation::Comment
- Seq parser: Bio::SeqIO

More Bioperl Objects

- BLAST, FASTA parsing: `Bio::SearchIO`
- Search objects: `Bio::Search::Result`,
`Bio::Search::Hit`, `Bio::Search::HSP`
- (old) BLAST parsing: `Bio::Tools::BPlite`
- MSA parsing: `Bio::AlignIO`
- MSA object: `Bio::SimpleAlign`

Bioperl Seq DB objects

- (old) Flatfile indexing: `Bio::Index::Fasta`, `Bio::Index::Swissprot`, `Bio::Index::EMBL`
- (new) Fasta only indexing: `Bio::DB::Fasta`
- (newest) OBDA flatfile standard: `Bio::DB::Flat`
- Remote DBs: `Bio::DB::GenBank`, `Bio::DB::EMBL`, `Bio::DB::Swissprot`

Other Bioperl Modules

- Bibliographic objects and Biblio DB access: Bio::Biblio
- PDB structures - Bio::Structure
- Unigene clusters - Bio::ClusterIO
- Coordinate remapping: Bio::Coordinate
- Render sequences, annotations: Bio::Graphics
- Gene prediction parsing: Bio::Tools::Genscan, Bio::Tools::Genemark, Bio::Tools::Genewise

Phylogenetic, Popgen, and MolEvo modules

- Phylogenetic trees: Bio::TreeIO, Bio::Tree
- Phylogenetic tools PHYLIP, PAML:
Bio::Tools::Phylo, Bio::Tools::Run::Phylo
- * Population Genetics: Bio::PopGen::Statistics
(Fst)
- Maps, Markers: Bio::Map
- * Pedigrees, Families, Genotypes:
Bio::Pedigree

Tools for running analyses

- `Bio::Tools::Run` provides framework for running analyses locally or remotely
- `Bio::Tools::Run::StandAloneBlast`,
`Bio::Tools::Run::RemoteBlast`
- Access to SOAP service analyses at EBI
- Access to CGI-based analyses at Pasteur via their PISE interface.

How to build your own modules

- Inherit from `Bio::Root::Root`
- Use `Bio::Root::IO` for file/fh input/output
- Follow the basic template for inheritance - chained constructors
- Reuse things like `Bio::Seq` for sequence data
- Follow POD guidelines - `bioperl.lisp` emacs macros make this easier.

Detailed look at Sequence parsing

- `Bio::SeqIO` is a factory
- `Bio::SeqIO::genbank`, `Bio::SeqIO::embl`, etc are driver modules which implement the `next_seq` and `write_seq` method
- Can produce `Bio::PrimarySeq`, `Bio::Seq`, or `Bio::Seq::RichSeq` depending on richness of the data

Look at the Sequence object

- Common (Bio::PrimarySeq) methods
 - seq() - get the sequence as a string
 - length() - get the sequence length
 - subseq(\$s,\$e) - get a subsequence
 - translate(...) - translate to protein [DNA]
 - revcom() - reverse complement [DNA]
 - display_id() - identifier string
 - description() - description string

Detailed look at Seqs with annotations

- Bio::Seq objects have the methods
 - add_SeqFeature(\$feature) - attach feature(s)
 - get_SeqFeatures() - get all the attached features.
 - species() - a Bio::Species object
 - annotation() - Bio::Annotation::Collection

Sequence Features

- `Bio::SeqFeatureI` - interface - GFF derived
 - `start()`, `end()`, `strand()` for location information
 - `location()` - `Bio::LocationI` object (to represent complex locations)
 - `score`, `frame`, `primary_tag`, `source_tag` - feature information
 - `spliced_seq()` - for attached sequence, get the sequence spliced.

Sequence Feature (cont.)

- Bio::SeqFeature::Generic
 - `add_tag_value($tag,$value)` - add a tag/value pair
 - `get_tag_value($tag)` - get all the values for this tag
 - `has_tag($tag)` - test if a tag exists
 - `get_all_tags()` - get all the tags

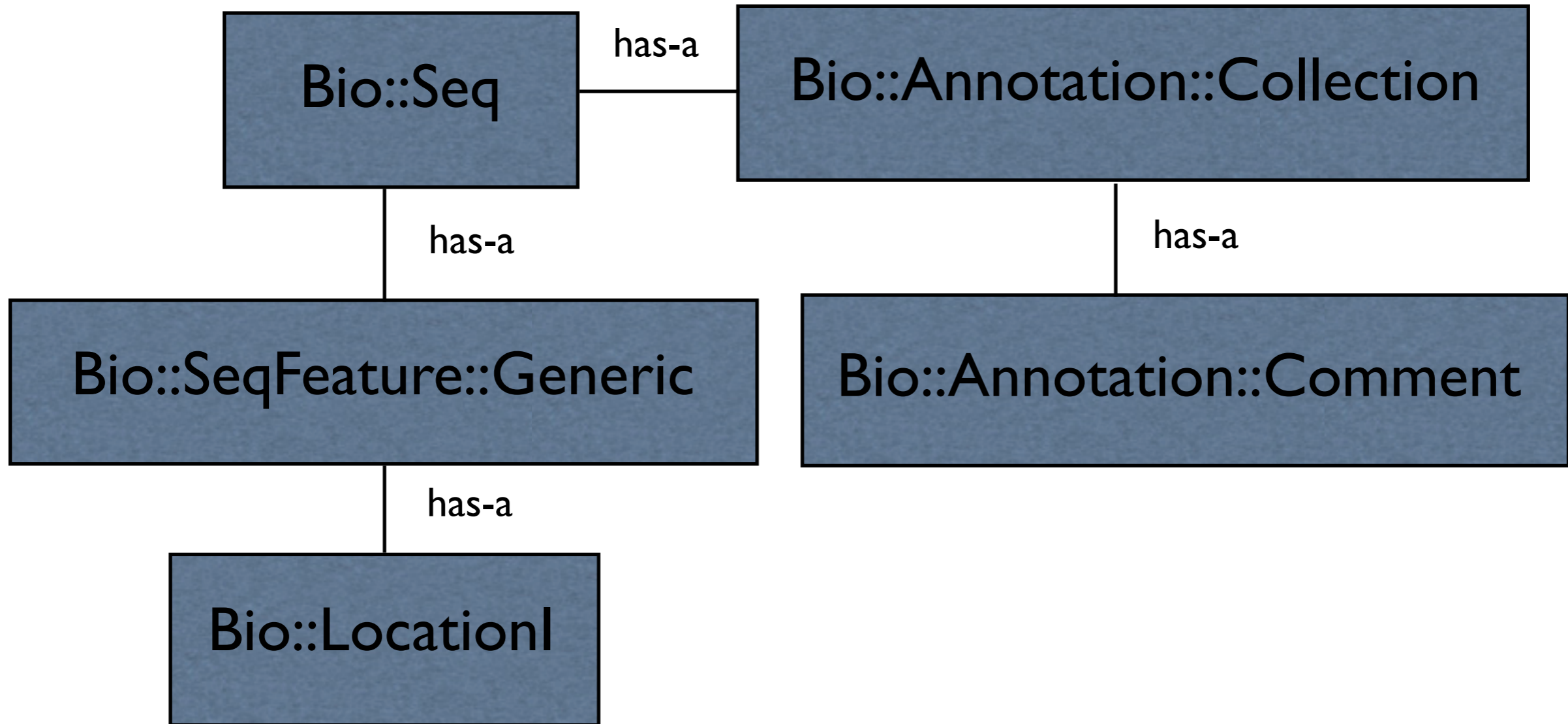
Annotations

- Each `Bio::Seq` has a `Bio::Annotation::Collection` via `$seq->annotation()`
- Annotations are stored with keys like 'comment' and 'reference'
- `@com=$annotation->get_Annotations('comment')`
- `$annotation->add_Annotation('comment', $an)`

Annotations

- `Annotation::Comment`
 - `comment` field
- `Annotation::Reference`
 - `author,journal,title, etc`
- `Annotation::DBLink`
 - `database,primary_id,optional_id,comment`
- `Annotation::SimpleValue`

Sequence & Annotation Schematic



Putting it all together

- Task
 - Parse SwissProt file
 - Print the Medline ID
 - Get the cross references to other seqs
 - Get the sequence of the gene which codes for this protein

SYNOPSIS

```
use Bio::SeqIO;

$in = Bio::SeqIO->new(-file => "inputfilename" , '-format' => 'Fasta');
$out = Bio::SeqIO->new(-file => ">outputfilename" , '-format' => 'EMBL');
# note: we quote -format to keep older Perls from complaining.

while ( my $seq = $in->next_seq() ) {
    $out->write_seq($seq);
}
```

Now, to actually get at the sequence object, use the standard Bio::Seq methods (look at [Bio::Seq](#) if you don't know what they are)

```
use Bio::SeqIO;

$in = Bio::SeqIO->new(-file => "inputfilename" , '-format' => 'genbank');

while ( my $seq = $in->next_seq() ) {
    print "Sequence ",$seq->id," first 10 bases ",$seq->subseq(1,10),"\n";
}
```

The SeqIO system does have a filehandle binding. Most people find this a little confusing, but it does mean you write the world's smallest reformatter

```
use Bio::SeqIO;

$in = Bio::SeqIO->newFh(-file => "inputfilename" , '-format' => 'Fasta');
$out = Bio::SeqIO->newFh('-format' => 'EMBL');

# World's shortest Fasta<->EMBL format converter:
print $out $_ while <$in>;
```

DESCRIPTION

Bio::SeqIO is a handler module for the formats in the SeqIO set (eg, Bio::SeqIO::fasta). It is the officially sanctioned way of getting at the format objects, which most people should use.

The Bio::SeqIO system can be thought of like biological file handles. They are attached to filehandles with smart formatting rules (eg, genbank format, or EMBL format, or binary trace file format) and can either read or write sequence objects (Bio::Seq objects, or more correctly, Bio::Seq implementing objects, of which Bio::Seq is one such object). If you want

Parse the SwissProt file

```
use Bio::SeqIO;  
  
my $in = Bio::SeqIO->new(-file => 'rod.sp',  
                        -format => 'swiss');  
  
my $seq = $in->next_seq;
```

NAME

Bio::Annotation::Collection - Default Perl implementation of AnnotationCollectionI

SYNOPSIS

```
# get an AnnotationCollectionI somehow, eg

$ac = $seq->annotation();

foreach $key ( $ac->get_all_annotation_keys() ) {
    @values = $ac->get_Annotations($key);
    foreach $value ( @values ) {
        # value is an Bio::AnnotationI, and defines a "as_text" method
        print "Annotation ",$key," stringified value ",$value->as_text,"\n";

        # also defined hash_tree method, which allows data orientated
        # access into this object
        $hash = $value->hash_tree();
    }
}
```

DESCRIPTION

Bioperl implementation for Bio::AnnotationCollectionI

FEEDBACK

Mailing Lists

User feedback is an integral part of the evolution of this and other Bioperl modules. Send your comments and suggestions preferably to one of the Bioperl mailing lists. Your participation is much appreciated.

```
bioperl-l@bioperl.org          - General discussion
http://bio.perl.org/MailList.html - About the mailing lists
```

Reporting Bugs

Report bugs to the Bioperl bug tracking system to help us keep track the bugs and their resolution. Bug reports can be submitted via email or the web:

```
bioperl-bugs@bioperl.org
http://bugzilla.bioperl.org/
```

AUTHOR - Ewan Birney

Print the Medline ID

```
my $ann = $seq->annotation();  
for my $ref ( $ann->get_Annotations('reference') ) {  
    print $ref->medline, "\n";  
}
```

Get the cross references to other seqs

```
my @xrefs;
for my $xref ( $ann->get_Annotations('dblink') ) {
    if( $xref->database() eq 'EMBL' ) {
        push @xrefs, $xref->primary_id;
    }
}
```

Retrieve these sequences

```
use Bio::DB::GenBank;
use Bio::DB::EMBL;
my $dbh = Bio::DB::GenBank->new();

foreach my $xrefid ( @xrefs ) {
    my $dbseq = $dbh->get_Seq_by_acc($xref);
    # check if it is DNA or mRNA
    print $dbseq->accession_number, " ",
          $xrefid, " ", $dbseq->molecule(), "\n";
}
```

MedlineID: 89342435 "Isolation of an active gene encoding human hnRNP protein A1. Evidence for alternative splicing."

MedlineID: 88233978 "cDNA cloning of human hnRNP protein A1 reveals the existence of multiple mRNA isoforms."

MedlineID: 87053868 "Mammalian single-stranded DNA binding protein UP I is derived from the hnRNP core protein A1."

MedlineID: 90214633 "Alternative splicing in the human gene for the core protein A1 generates another hnRNP protein."

MedlineID: 95247808 "A nuclear localization domain in the hnRNP A1 protein."

MedlineID: 96067639 "A nuclear export signal in hnRNP A1: a signal-mediated, temperature-dependent nuclear protein export pathway."

MedlineID: 95286702 "Nucleo-cytoplasmic distribution of human hnRNP proteins: a search for the targeting domains in hnRNP A1."

MedlineID: 91099515 "Modeling by homology of RNA binding domain in A1 hnRNP protein."

MedlineID: 97307256 "Crystal structure of the two RNA binding domains of human hnRNP A1 at 1.75-Å resolution."

MedlineID: 97277240 "Crystal structure of human UP1, the domain of hnRNP A1 that contains two RNA-recognition motifs."

X12671 X12671 DNA

X06747 X06747 mRNA

X04347 X04347 mRNA

X79536 X79536 mRNA

A Detailed look at BLAST parsing

- 3 Components
 - Result: Bio::Search::Result::ResultI
 - Hit: Bio::Search::Hit::HitI
 - HSP: Bio::Search::HSP::HSPi

Using the Search::Result object

```
use Bio::SearchIO;
use strict;
my $parser = new Bio::SearchIO(-format => 'blast', -file => 'file.bls');
while( my $result = $parser->next_result ){
    print "query name=", $result->query_name, " desc=",
          $result->query_description, ", len=", $result->query_length, "\n";
    print "algorithm=", $result->algorithm, "\n";
    print "db name=", $result->database_name, " #lets=",
          $result->database_letters, " #seqs=", $result->database_entries, "\n";
    print "available params ", join(',',
          $result->available_parameters), "\n";
    print "available stats ", join(',',
          $result->available_statistics), "\n";
    print "num of hits ", $result->num_hits, "\n";
}
```

Using the Search::Hit Object

```
use Bio::SearchIO;
use strict;
my $parser = new Bio::SearchIO(-format => 'blast', -file => 'file.bls');
while( my $result = $parser->next_result ){
    while( my $hit = $result->next_hit ) {
        print "hit name=", $hit->name, " desc=", $hit->description,
            "\n len=", $hit->length, " acc=", $hit->accession, "\n";
        print "raw score ", $hit->raw_score, " bits ", $hit->bits,
            " significance/evaluate=", $hit->evaluate, "\n";
    }
}
```

Cool Hit Methods

- `start()`, `end()` - get overall alignment start and end for all HSPs
- `strand()` - get best overall alignment strand
- `matches()` - get total number of matches across entire set of HSPs (can specify only exact 'id' or conservative 'cons')

Using the Search::HSP Object

```
use Bio::SearchIO;
use strict;
my $parser = new Bio::SearchIO(-format => 'blast', -file => 'file.bls');
while( my $result = $parser->next_result ){
    while( my $hit = $result->next_hit ) {
        while( my $hsp = $hit->next_hsp ) {
            print "hsp evaluate=", $hsp->evaluate, " score=" $hsp->score, "\n";
            print "total length=", $hsp->hsp_length, " qlen=",
                $hsp->query->length, " hlen=", $hsp->hit->length, "\n";
            print "qstart=", $hsp->query->start, " qend=", $hsp->query->end,
                " qstrand=", $hsp->query->strand, "\n";
            print "hstart=", $hsp->hit->start, " hend=", $hsp->hit->end,
                " hstrand=", $hsp->hit->strand, "\n";
            print "percent identical ", $hsp->percent_identity,
                " frac conserved ", $hsp->frac_conserved(), "\n";
            print "num query gaps ", $hsp->gaps('query'), "\n";
            print "hit str =", $hsp->hit_string, "\n";
            print "query str =", $hsp->query_string, "\n";
            print "homolog str=", $hsp->homology_string, "\n";
        }
    }
}
```

Cool HSP methods

- rank() - order in the alignment (which you could have requested, by score, size)
- matches
- seq_inds - get a list of numbers representing residue positions which are
- conserved, identical, mismatches, gaps

Turning BLAST into HTML

```
use Bio::SearchIO;
use Bio::SearchIO::Writer::HTMLResultWriter;

my $in = new Bio::SearchIO(-format => 'blast',
                          -file   => shift @ARGV);

    my $writer = new
Bio::SearchIO::Writer::HTMLResultWriter();
    my $out = new Bio::SearchIO(-writer => $writer
                              -file   => ">file.html");
    $out->write_result($in->next_result);
```

Turning BLAST into HTML

```
# to filter your output
my $MinLength = 100; # need a variable with scope outside the method
sub hsp_filter {
    my $hsp = shift;
    return 1 if $hsp->length('total') > $MinLength;
}
sub result_filter {
    my $result = shift;
    return $hsp->num_hits > 0;
}

my $writer = new Bio::SearchIO::Writer::HTMLResultWriter
    (-filters => { 'HSP' => \&hsp_filter } );
my $out = new Bio::SearchIO(-writer => $writer);
$out->write_result($in->next_result);

# can also set the filter via the writer object
$writer->filter('RESULT', \&result_filter);
```

Custom URL links

```
@args = ( -nucleotide_url => $gbrowsedblink,
          -protein_url    => $gbrowsedblink
        );
my $processor = new
Bio::SearchIO::Writer::HTMLResultWriter(@args);
$processor->introduction(\&intro_with_overview);
$processor->hit_link_desc(\&gbrowse_link_desc);
$processor->hit_link_align(\&gbrowse_link_desc);

sub intro_with_overview {
    my ($result) = @_ ;
    my $f = &generate_overview($result,$result->{"_FILEBASE"});
    $result->rewind();
    return sprintf(
    qq{
    <center>
    <b>Hit Overview<br>
    Score: <font color="red">Red=47(&gt;=200)</font>, <font
    color="purple">Purple 200-80</font>, <font color="green">Green
```

Running BLAST Remotely

- Allow submission of BLAST queries to NCBI via scripts
- Need to be careful - infinite loops, over submitting jobs can get your access shutdown!

```

use Bio::Tools::Run::RemoteBlast;
my $prog = 'blastp';
my $db    = 'ecoli';
my $e_val= '1e-10';
my $remote_blast = Bio::Tools::Run::RemoteBlast->new(
    -prog    => $prog,
    -data    => $db,
    -expect => $e_val);
my $r = $remote_blast->submit_blast($inputfilename);
while( my @rids = $remote_blast->each_rid ) {
    for my $rid ( @rids ) {
        my $rc = $remote_blast->retrieve_blast($rid);
        if( ! ref($rc) ) {
            if( $rc < 0 ) { $remote_blast->remove_rid($rid); }
            print STDERR "."; sleep(10);
        } else {
            $remote_blast->remove_rid($rid);
            my $result = $rc->next_result;
            while( my $hit = $result->next_hit ) {
                print $hit->name, " ", $hit->significance, "\n";
            }
        }
    }
}

```

BLAST 2 GFF

- Let's turn a BLAST report into GFF format
- This can be loaded into Gbrowse (later in the course) and visualized

BLAST 2 GFF

```
use Bio::SearchIO;
use Bio::Tools::GFF; # <-- operates on SeqFeatures
my $in = new Bio::SearchIO( -file => $file,
                           -format => 'blast' );

my $out = new Bio::Tools::GFF;
while( my $r = $in->next_result ) {
    while( my $hit = $r->next_result ) {
        while( my $hsp = $hit->next_hsp ) {
            $out->write_feature($hsp->hit);
        }
    }
}
```