Current Issues In Perl Programming

Lukas Thiemeier
Current Issues In Perl Programming
DESY, Zeuthen, 2011-04-26

Overview

- **Introduction**
- Moose – modern object-orientation in Perl
- DBIx::Class – comfortable an flexible database access
- Catalyst – a MVC web application framework

Introduction

- What is this talk about?
  - Modern Perl can do more than most people know
  - A **quick** overview about **some** modern features
  - Illustrated with some short examples
- What is this talk **not** about?
  - Not an introduction to the Perl programming language
  - Not a Perl tutorial
  - Not a complete list of all current issues in Perl 5
  - Not a complete HowTo for the covered topics

Overview

- Introduction
- **Moose – modern object-orientation in Perl**
  - About Moose
  - Creating and extending classes
  - Some advanced features
- DBIx::Class – comfortable an flexible database access
- Catalyst – a MVC web application framework

About Moose

- "A postmodern object system for Perl 5"
- Based on Class::MOP, a metaclass system for Perl 5
- Look and feel similar to the Perl 6 object syntax

"The main goal of Moose is to make Perl 5 Object Oriented programming easier, more consistent and less tedious. With Moose you can to think more about what you want to do and less about the mechanics of OOP."

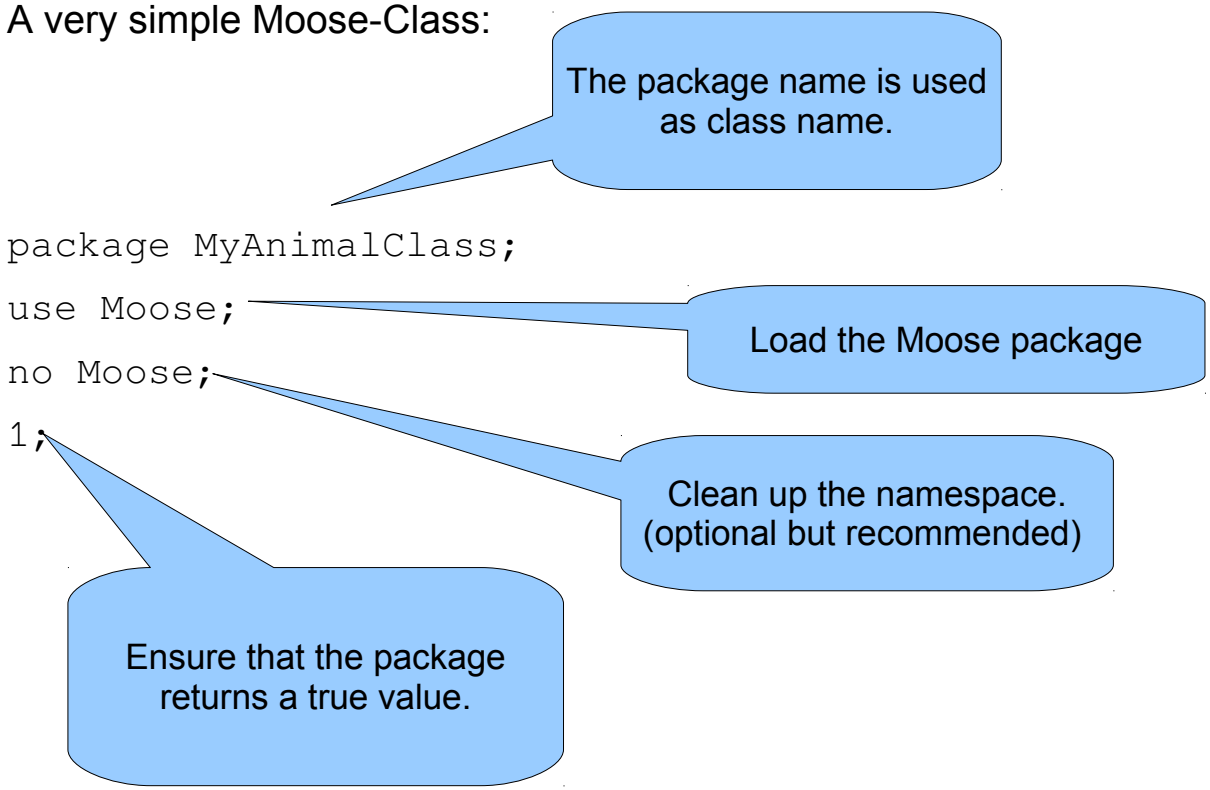
Creating Classes

- A very simple Moose-Class:
  - Create a file called "MyAnimalClass.pm" with the following content:

```
package MyAnimalClass;
use Moose;
no Moose;
1;
```


Creating Classes

- A very simple Moose-Class:



Creating Objects

- The previous example is completely functional
- Moose creates the class, including a constructor, automatically
- Objects of the class can be created by calling the constructor:

```
use MyAnimalClass;
my $new_simple_object = MyAnimalClass->new();
```

Creating Objects

- T ompletely functional
- M cluding a
- Objects of the cl n be created b

Importing “MyAnimalClass” to the current namespace

We are progrmamming in Perl, and the constructor is an ordinary subroutine, so the brackets are optional.

```
use MyAnimalClass;
my $new_simple_object = MyAnimalClass->new();
```

Initializing a private variable which stores the new object.

The object is created by calling the classes constructor.

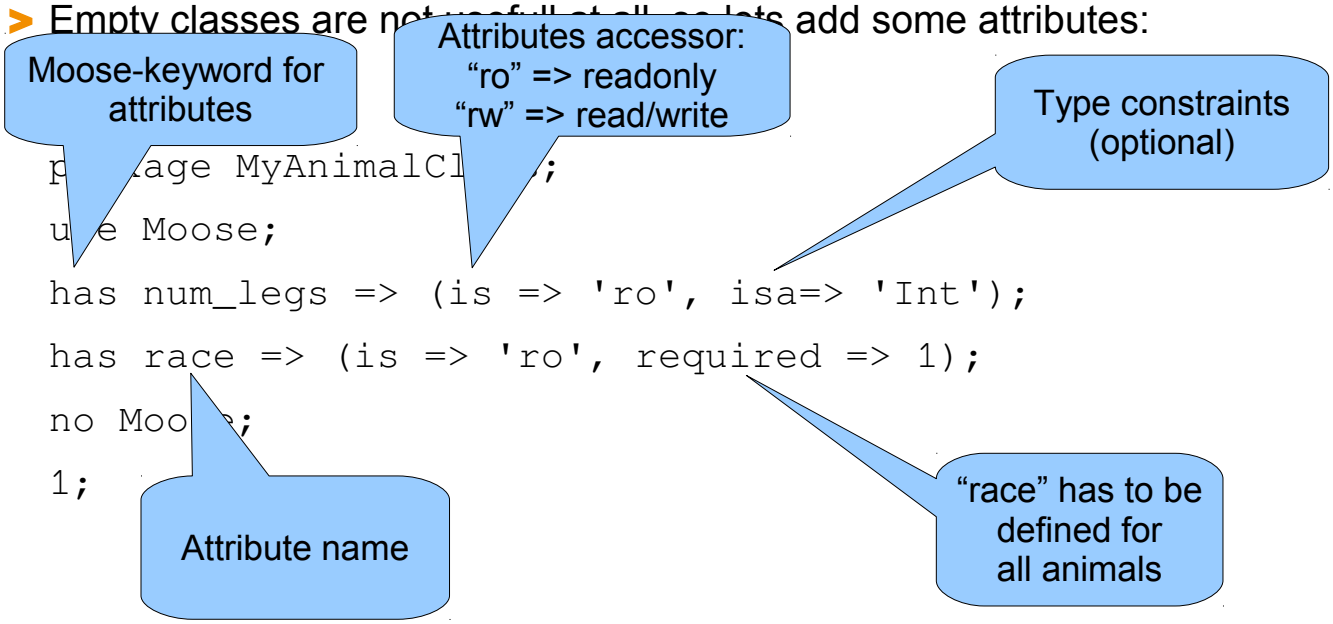
Defining Attributes

- Empty classes are not usefull at all, so lets add some attributes:

```
package MyAnimalClass;
use Moose;
has num_legs => (is => 'ro', isa=> 'Int');
has race => (is => 'ro', required => 1);
no Moose;
1;
```


Defining Attributes

- Empty classes are not useful at all, so lets add some attributes:



Creating/Using Objects – a valid example

- Moose creates the accessors, getter- and setter-methods, and performs validation automatically

```
use MyAnimalClass;

my $dog = MyAnimalClass->new(
   race => 'Dog',
   num_legs => 4
);
print  “a  $dog->race has $dog->num_legs legs.\n”;
```

Creating/Using Objects – an invalid example

```
Use MyAnimalClass;
my $twolegg = MyAnimalClass->new(num_legs =>2);

my $horse = MyAnimalClass->new(race => 'Horse',
num_legs => 'green');

my $dog = MyAnimalClass->new(race => 'Dog', num_legs
=> 4);
$dog->num_legs(2);
```


Creating/Using Objects – an invalid example

```
Use MyAnimalClass;
my $twolegg = MyAnimalClass->new(num_legs =>2);
```

Missing 'race'-attribute

```
my $horse = MyAnimalClass->new(race => 'Horse',
num_legs => 'green');
```

Violating type constraints ('green' is not an Int)

Valid

```
my $dog = MyAnimalClass->new(race => 'Dog', num_legs
=> 4);
$dog->num_legs(2);
```

Invalid: writing a readonly attribute

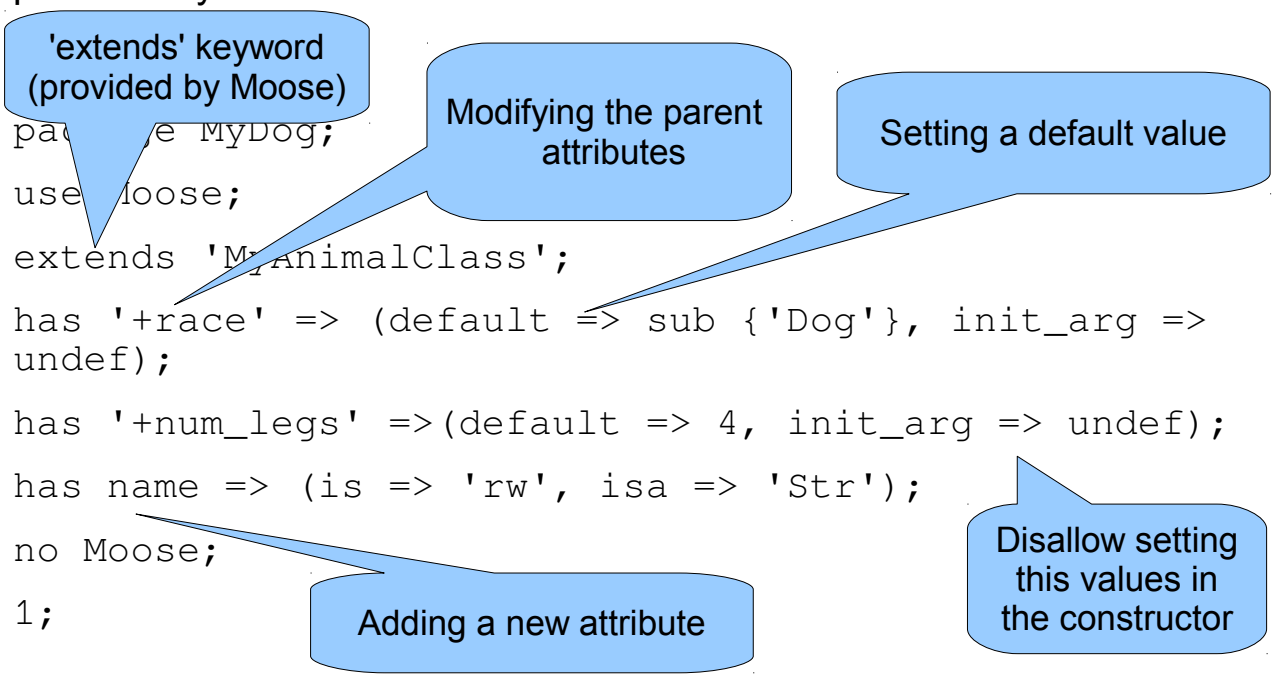
Extending Classes

- One key feature of object orientated programming is inheritance, which is provided by Moose as well:

```
package MyDog;
use Moose;
extends 'MyAnimalClass';
has '+race' => (default => sub {'Dog'}, init_arg =>
undef);
has '+num_legs' =>(default => 4, init_arg => undef);
has name => (is => 'rw', isa => 'Str');
no Moose;
1;
```


Extending Classes

- One key feature of object orientated programming is inheritance, which is provided by Moose aswell:



Some Advanced Features - TypeConstraints

- Standart types in Perl: scalar, list, hash and reference
- Moose introduces several type constraints like
  - Bool
  - Str
  - Int
  - ...
- Moose makes it possible to
  - Verify type constraints automatically
  - Extend existing typeconstraints
  - Define new typeconstraints
  - Perform coercion
- Moose **does not** provide a new type system for Perl5

TypeConstraints – an example

```
package MyAnimalClass;
...

use Moose::Util::TypeConstraints;

subtype 'EvenInt'
   => as 'Int'
   => where { $_ % 2 == 0};
...

has num_legs => (is => 'ro', isa=> 'EvenInt');
...
```


TypeConstraints – an example

```
package MyAnimalClass;
...
use Moose::Util::TypeConstraints;
subtype 'EvenInt'
   => as 'Int'
   => where { $_ % 2 == 0};
...
has num_legs => (is => 'ro', isa=> 'EvenInt');
...
```

Importing TypeCosntraint Utilities

Keyword

Name of the new constraint

Parent type constraint

condition

Using the new constraint

Some Advanced Features – method modifiers

- Method modifiers allow the programmer to influnce how methods behave, without changing the methods code
- Moose provides five method modifiers:
  - **before** – do something BEFORE a certain method is executed
  - **after** – do something AFTER a certain method was executed
  - **around** – modifies am methods parameters and/or return value
  - **override** – override a method defined in a parent class, allows calling the overridden method within the override block.
  - **augment** - “reverse override”

Method Modifiers – an example

```
package MyDog;
...
after 'race' => sub{
   my $self = shift;
   warn “the 'race'-accessor was called in 'MyDog'”;
};
...
```


Moose – more points of interest

- The **MooseX** namespace provides extensions to Moose, for example:
  - MooseX::NonMoose, MooseX::Alien, MooseX::InsideOut – extending non-Moose Classes with Moose
  - MooseX::Declare – A more declarative way to define Classes with Moose
- Moose::Role
  - Something between “interfaces” and “abstract classes” in Java
  - Roles can require the presence of certain methods in classes that 'do' the role
  - Roles can increase modularity and reusability of the code
  - Roles can be applied to objects at runtime

QUESTIONS?

Overview

- Introduction
- Moose – modern object-orientation in Perl
- **DBIx::Class – comfortable an flexible database access**
  - About DBIx::Class
  - Some core modules
  - A simple example
- Catalyst – a MVC web application framework

About DBIx::Class

- An **O**bject – **R**elational – **M**apper for Perl 5
- Backends are available for:
  - MySql
  - SqLite3
  - PostgreSQL
  - Oracle
  - …
- Modular structure
- Highly configurable
- Highly extensible
- Easy to use

About DBIx::Class

- DBIx::Class allows:
  - Creation and alteration of tables
  - Database conversion (e.g. MySQL → SQLite)
  - Database usage (insert, select, delete, update)
  - Additional features
- In an object oriented way
- Without writing a single SQL statement

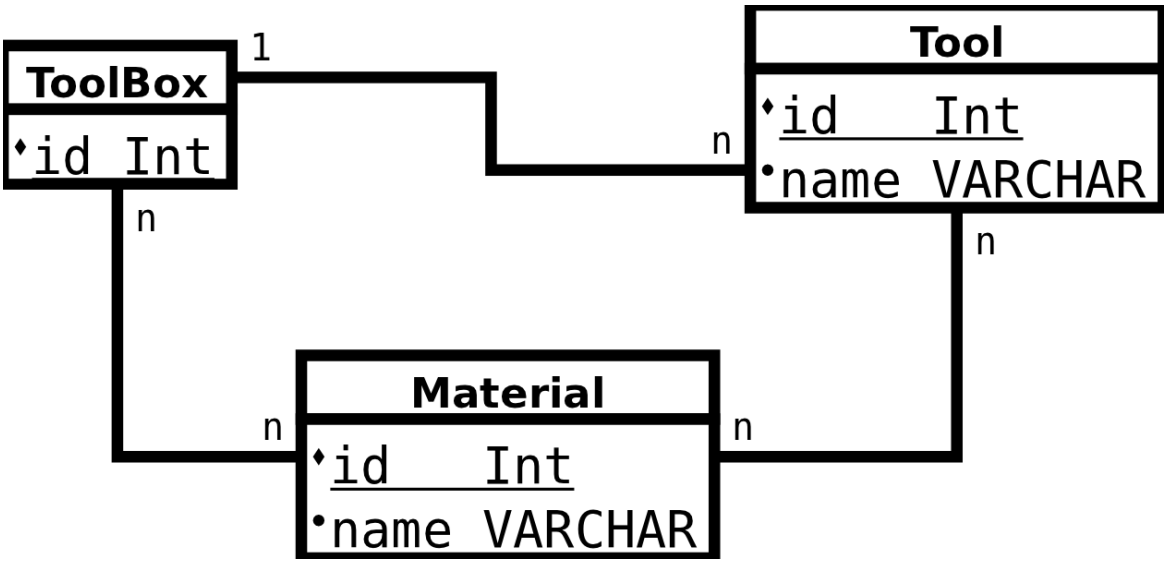
About DBIx::Class

- Further points of interest:
  - Support for static and dynamic database schemas
  - Supports creation of a schema for a given database
  - Supports creation of a database for a given schema

Some Core Modules in DBIx::Class (simplified)

- DBIx::Class::Schema
  - The core of the database schema
  - Connect to DB
  - Authentication
  - …
- DBIx::Class::Row
  - A single row
- DBIx::Class::ResultSet
  - A collection of Rows
  - Searching and inserting data
  - Allowes chained queries

DBIx::Class Example – database layout



DBIx::Class Example – files (simplified)

- The Schema Class
  - lib/ToolBoxDB/Schema.pm
- The Row Classes
  - lib/ToolBoxDB/Schema/Result/ToolBox.pm
  - lib/ToolBoxDB/Schema/Result/Tool.pm
  - lib/ToolBoxDB/Schema/Result/Material.pm
  - lib/ToolBoxDB/Schema/Result/ToolBoxMaterial.pm
  - lib/ToolBoxDB/Schema/Result/ToolMaterial.pm

DBIx::Class::Schema Example – the schema class

```
package ToolBoxDB::Schema;

use strict;
use warnings;
use base 'DBIx::Class::Schema';
__PACKAGE__->load_namespaces;
1;
```


DBIx::Class::Schema Example – the schema class

```
package ToolBoxDB::Schema;

use strict;
use warnings;
use base 'DBIx::Class::Schema';
__PACKAGE__->load_namespaces;
1;
```

Name of the schema class

Extending 'DBIx::Class::Schema' (not using Moose)

Automatically load packages in “ToolBoxDB::Schema::*”

DBIx::Class::Schema Example – ToolBox Result Class

```
package ToolBoxDB::Schema::Result::ToolBox;
use strict;
use warnings;
use base 'DBIx::Class::Core';
__PACKAGE__->table('toolbox');
__PACKAGE__->add_columns(
  "id", {
    data_type => "INTEGER",
    is_nullable => 0,
    is_auto_increment => 1,
  },);
__PACKAGE__->set_primary_key("id");
...
```


ToolBox ResultClass (continued)

```
...
__PACKAGE__->has_many(
"toolbox_materials",
"ToolBoxDB::Schema::Result::ToolBoxMaterial",
{ "foreign.toolbox_id" => "self.id" },
{ cascade_copy => 1, cascade_delete => 1 },
);
__PACKAGE__->many_to_many(
"materials",toolbox_materials => 'material');
...
1;
```


DBIx::Class::Schema Example – ToolBox Result Class

```
package ToolBoxDB::Schema::Result::ToolBox;
use strict;
use warnings;
use base 'DBIx::Class::Core';
__PACKAGE__->table('toolbox');
__PACKAGE__->add_columns(
  "id", {
    data_type => "INTEGER",
    is_nullable => 0,
    is_auto_increment => 1,
  },);
__PACKAGE__->set_primary_key("id");
...
```

Extends DBIx::Class::Row and others

Table name

Column definitions

Primary key

ToolBox ResultClass (continued)

```
...
__PACKAGE__->has_many(
"toolbox_materials",
"ToolBoxDB::Schema::Result::ToolBoxMaterial",
{ "foreign.toolbox_id" => "self.id" },
{ cascade_copy => 1, cascade_delete => 1 },
);
__PACKAGE__->many_to_many(
"materials",toolbox_materials => 'material');
...
1;
```

1:n relationship

n:m relationship

DBIx::Class Example: Connecting to the Database

```
use ToolBoxDB::Schema;

my $dsn = 'dbi:SQLite:dbname=db/toolbox.db';
my $user = "";
my $pass = "";

my $schema = ToolBoxDB::Schema->connect(
    $dsn,
    $user,
    $pass,
    {}
);
```


DBIx::Class Example: Connecting to the Database

```
use ToolBoxDB::Schema;
my $dsn = 'dbi:SQLite:dbname=db/toolbox.db';
my $user = "";
my $pass = "";

my $schema = ToolBoxDB::Schema->connect(
    $dsn,
    $user,
    $pass,
    {}
);
```

Importing schema

Select backend

Username and password

Database name

Storing the schema for further use

Calling the schemas connect-method

Additional attributes

DBIx::Class Example: Using the Database

```
…
my $schema = ToolBoxDB::Schema->connect( $dsn, $user,
$pass, {} );
my $toolbox_id = shift;
my $toolbox =  $schema->resultset("ToolBox")->find(
    $toolbox_id
);
die "no such ToolBox $toolbox_id" unless $toolbox;
my $toolcount = $toolbox->tools->count;
print "ToolBox $toolbox_id has $toolcount Tools\n";
```


DBIx::Class Example: Using the Database

```
…
my                BoxDB::Schema->connect( $dsn, $user,
$pa
my $toolbox_id = shift;
my $toolbox =  $schema->resultset("ToolBox")->find(
    $toolbox_id
);
die "no such ToolBox $toolbox_id" unless $toolbox;
my $toolcount = $toolbox->tools->count;
print "ToolBox $toolbox_id has $toolcount Tools\n";
```

Using first paramete As toolbox id

Selecting table

Searching by toolbox id

Calling “tools” accessor

Calling “count” method In “Tool”-ResultSet

Overview

- Introduction
- Moose – modern object-orientation in Perl
- DBIx::Class – comfortable an flexible database access
- **Catalyst – a MVC web application framework**
  - About Catalyst
  - About MVC
  - Creating Catalyst Applications

About Catalyst

- A “Model – View – Controller” Web Application Framework for Perl5
- Uses Moose as object system
- Supports DBIx::Class based database models

“Catalyst is a modern framework for making web applications without the pain usually associated with this process.”

About Catalyst

Catalyst is divided into three main packages:

- Catalyst::Runtime
  - Needed to run Catalyst applications
- Catalyst::Devel
  - Tools and helpers needed to create Catalyst applications
- Catalyst::Manual
  - A large collection of tutorials and other documentation

About MVC (simplified)

- Model – View – Controller (MVC) is an architectural pattern in software engineering
- The application is divided into three parts:
  - **The Model** is responsible for all data-related actions
  - **The View** interacts with the user by displaying all kind of output and allowing the user to provide some input
  - **The Controller** receives and analyses the input, delegates it to the model (if necessary) and calls the View to render the next output
- Benefits
  - Model, View and Controller can be developed and tested independent from each other
  - The modular structure eases code reuse

Creating A Catalyst Application

- With Catalyst::Devel installed, creating a new application is as easy as running

```
# catalyst.pl MyNewApp
```

- This creates a new directory “MyNewApp”, containting
  - A application core module called lib/MyNewApp.pm
  - Some basic testcases in t/
  - A Makefile-generator for easy deployment and testing
  - Several helper scripts for
    - Creating new Controllers, Views and Models
    - Running the automated tests
    - Starting a testserver
    - A fastcgi-server (for use with a standalone webserver)

Creating A Model

- Once the new application is created, models, views and controllers can be added by calling the generated helper scripts:

```
# MyNewApp/script/mynewapp_create.pl \
      model MyDB DBIC::Schema MyNewApp::Schema \
      create=static dbi:SQLite:dbname=my.db
```


Creating A Model

- Once the new application is created, m ers can be added by calling the generated help

Calling the generated create-script

```
# MyNewApp/script/mynewapp_create.pl \
      model MyDB DBIC::Schema MyNewApp::Schema \
      create=static dbi:SQLite:dbname=my.db
```

Creating model called MyDB

We are creating a DBIC model

Creating A Model

- Once the new application is created, models, views and controllers can be added by calling the generated helper scripts:

```
# MyNewApp/script/mynewapp_create.pl \
      model MyDB DBIC::Schema MyNewApp::Schema \
      create=static dbi:SQLite:dbname=my.db
```

Database configuration

Base name of schema-classes

Creating Catalyst Applications

- A quick example with Catalyst::Example::InstantCRUD

QUESTIONS?

References

- http://search.cpan.org/dist/Catalyst-Manual/lib/Catalyst/Manual.pm
- http://search.cpan.org/~doy/Moose-2.0002/lib/Moose/Manual.pod
- http://search.cpan.org/~frew/DBIx-Class-0.08191/lib/DBIx/Class.pm
- Kieren Diment and Matt S Trout – The Definite Guide to Catalyst
  - ISBN978-1-4302-2365-8