Lesson Number 9

Name:

Introducing PHP ActiveRecord

Description:

GROUP ACTIVITY - PHP Classes Recap

Objective

To demonstrate an understanding of classes.

Instructions

1. Get into groups of 2 or more
2. Write a class that will generate an HTML select field from a passed argument
3. Demonstrate your class by initializing it and outputting the select field on an HTML page

Challenges:

- What parameters will you need?
- What datatypes will you support (string, array, integer, boolean, etc...)?
- Are any of the parameters 'optional'?
- Does your class allow for other HTML methods to be made? Or is it strictly for select fields only?
- Does your class have a need for a constructor method?
- Does your class have need for properties?

Requirements:

- All properties must be private. (You will need getter/setter methods if you need to access them from the instantiated object.)
- The whole group should participate.
- The presentation should answer the following questions:
    1. Is you class strictly for building select fields?
    2. What challenges did you incur when planning your class?
    3. If you were to change anything about your class, what would it be?

ORM - Object Relational Mapping

What is it?

An ORM Framework is software that wraps your database in classes, allowing you to interact utilizing methods and properties instead of SQL.

Why do I want to use it?

ORMs allow for abstraction between your application and your datastore, as the ORM bridges the communication. You don't have to be aware of the database (or datastore) your using, as all CRUD interactions are done through the ORM.

ORM frameworks make CRUD operations simple and fast, and tend to be easier to learn as they only require an understanding of the data you're storing, not the syntax of the SQL database you're using.

How available are ORM Frameworks?

Most popular languages have ORM frameworks available for them. In addition, many development frameworks such as Laravel, Rails, and CakePHP have ORM frameworks built into their framework.

Active Record Pattern

- PHP Active Record is a framework that utilizes the active-record database pattern
- The pattern is a simple data-access method that maps a database table into an object
- this helps to eliminate duplication with code
- it also centralizes business logic
- PHP has other ORMs that utilize this pattern including Eloquent and Propel
- MVC frameworks generally implement this same pattern as well, such as Laravel, Symphony, and Rails
- there are active-record frameworks for most languages including .NET and Java

ACTIVITY - Introducing ActiveRecord for PHP

PHPActiveRecord Library

Configuration:

1. Download/clone the library from https://github.com/jpfuentes2/php-activerecord to /comp-1006/lesson-09/vendors/
2. Rename the folder to php-activerecord
3. Create a new file in /comp-1006/lesson-09/examples/ called config.php
4. Your config file should look like the following

```
<?php

                require_once $_SERVER['DOCUMENT_ROOT'] . '/lesson-
09/examples/vendors/php-activerecord/ActiveRecord.php';

                ActiveRecord\Config::initialize( function( $cfg ) {
                  $cfg->set_model_directory( $_SERVER['DOCUMENT_ROOT']
. '/lesson-09/examples/models' );
                  $cfg->set_connections(
                    array(
                      'development' =>
'mysql://root:root@localhost/comp-1006-lesson-examples'
                    )
                  );
                });
```

5. Replace the following line 'mysql://root:root@localhost/comp-1006-lesson-examples' with your details:
   'mysql://username:password@host/dbname'
6. This file must be included at the top of any script interacting with the database

Models:

Models represent your data and any rules associated with it. In the model, you will define the relationship your data object has with other data objects. In addition, you will define any validation and sanitization of data that is needed. You will also create methods to mutate the data into commonly used formats.

1. Create a new folder in /comp-1006/lesson-09/examples/ titled 'models' (plural)
2. Inside /comp-1006/lesson-09/examples/models/, create a new file titled category.php (singular)
3. Inside /comp-1006/lesson-09/examples/models/, create a new file titled product.php (singular)

PHP ActiveRecord requires models to be named in their singular form. PHP ActiveRecord has a built in reference library to associate singular and plural for the majority of words.

1. Open category.php and product.php in your IDE
2. Add the following code to /comp-1006/lesson-09/examples/models/category.php

```
<?php

                class Category extends ActiveRecord\Model {

                }
```

3. Add the following code to /comp-1006/lesson-09/examples/models/category.php

```
<?php

                class Product extends ActiveRecord/Model {

                }
```

If your database has relationships set up, you must define these in your models as associations. For example, if a Category can have many Products, you define this in the Category Model:

```
<?php

                class Category extends ActiveRecord\Model {
                  static $has_many = array( 'products' );

                }
```

It is safe to say that if Category has many products, Product obviously belongs to category.

```
<?php

                class Product extends ActiveRecord\Model {

                  static $belongs_to = array( 'category' );

                }
```

You may notice that the static properties $has_many and $belongs_to are assigned an array. This is because PHP ActiveRecord will read the array and extract any configuration options that have been set. By default, we only need to pass the relationship table in either the singular or plural form. You can see that common sense has been applied here: Category has_many products. Product belongs_to category.

These associations create some wonderful magic in the background allowing you to access the associated table's data easily without creating complex joins.

Naming Conventions:

So far you have seen some of PHP ActiveRecord's naming conventions. Singular and Plural defined values are important as they have context to ActiveRecord. Here are some naming convention rules:

1. Tables in the database must always be named in plural form. Ie: (parents, children, categories, products)
2. Models must always be named with the singular form of your table name. Ie: (parent, child, category, product)
3. The primary key in the database table must always be 'id'
4. In models, some methods require specific naming conventions to get them to execute at the appropriate time. get_ and set_ are two such methods that you will utilize often
   IE: get_first_name(), get_last_name(), set_first_name($value), set_last_name($value)

In our two models we will define some get_ and set_ methods so you can see these in operation. The great thing about these methods is it allows us to sanitize data going in to our database and coming out to our screen.

/comp-1006/lesson-09/examples/models/category.php

```
<?php

                class Category extends ActiveRecord\Model {

                  $has_many( array( 'products' ) );

                  public function set_name ( $name ) {
                    $this->assign_attribute( 'name', filter_var( $name,
FILTER_SANITIZE_STRING ) );
                  }

                  public function get_name () {
                    return htmlentities( $this->name );
                  }

                }
```

/comp-1006/lesson-09/examples/models/product.php

```
<?php

                class Product extends ActiveRecord\Model {

                  static $belongs_to = array( 'category' );

                  public function set_name ( $name ) {
                    $this->assign_attribute( 'name', filter_var( $name,
FILTER_SANITIZE_STRING );
                  }

                  public function set_price ( $price ) {
                    $this->assign_attribute( 'price', filter_var( $price,
FILTER_SANITIZE_NUMBER_FLOAT ) );
                  }

                  public function get_name () {
                    return htmlentities( $this->name );
                  }

                  public function get_price () {
                    return number_format( $this->price, 2, '.' );
                  }

                }
```

Validations

Validations are handled within the model. PHP ActiveRecord ships with several validation methods that you can use, as well as custom validations.

Let's add some validation to our models.

/comp-1006/lesson-09/examples/models/category.php

```
<?php

                class Category extends ActiveRecord\Model {
                  // associations
                  static $has_many = array( 'products' );

                  // setters
                  public function set_name ( $name ) {
                    $this->assign_attribute( 'name', filter_var( $name,
FILTER_SANITIZE_STRING ) );
                  }

                  // getters
                  public function get_name () {
                    return htmlentities( $this->read_attribute( 'name' ) );
                  }

                  // validations
                  static $validates_presence_of = array(
                    array( 'name', 'message' => 'must be present' )
                  );

                  static $validates_uniqueness_of = array(
                    array( 'name', 'message' => 'this category already
exists' )
                  );

                }
```

/comp-1006/lesson-09/examples/models/product.php

```
<?php

                class Product extends ActiveRecord\Model {

                  // associations
                  static $belongs_to = array( 'category' );

                  // setters
                  public function set_name ( $name ) {
                    $this->assign_attribute( 'name', filter_var( $name,
FILTER_SANITIZE_STRING ) );
                  }

                  public function set_price ( $price ) {
                    $this->assign_attribute( 'price', filter_var( $price,
FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION ) );
                  }

                  // getters
                  public function get_name () {
                    return htmlentities( $this->read_attribute( 'name' ) );
                  }
                  // create a getter to pull formatted prices
                  public function get_price_formatted () {
                    return '$' . number_format( $this->read_attribute(
'price' ), 2, '.', ',' );
                  }

                  // validations
                  static $validates_presence_of = array(
                    array( 'name', 'message' => 'must be present' ),
                    array( 'price', 'message' => 'must be present' )
                  );

                  static $validates_uniqueness_of = array(
                    array(
                      array( 'name', 'category_id' ),
                      'message' => 'this product already exists for this
category'
                    )
                  );

                  static $validates_numeralicity_of = array(
                    array( 'price', 'greater_than' => 0.01, 'message' =>
'must be greater than 1 cent' )
                  );

                  public function validate () {
                    // validate the presence of the category
                    if ( !Category::exists( $this->category_id ) ) $this-
>errors->add( 'category_id', "the selected category doesn't exist" );
                  }

                }
```

$validates_presence_of requires the attribute to be set, otherwise the message will be assigned to the errors property.

$validates_uniqueness_of triggers a check against the provided attribute to see if it alreay exists. If it does, the message will be assigned to an errors property.