asgn04 Constructors

Video

Watch chapter 6: Magic Methods from the PHP: Object-oriented Programming video from InLearning.

Setup

- Create a new folder named `asgn04-constructors` in your `web250` folder.
- Create a file named `constructor.php` inside your `asgn04-constructors` folder.

Git

- Stage and commit
- Create a new branch named `asgn04-constructor`
- Switch to the new branch

Basic constructor

This exercise is based off video chapter 6.1, *Contstructor Method*

- Create a class named Bird with two public properties
  - `commonName`
  - `latinName`

- Note: when creating a constructor you must use the `public` modifier.
- Create a constructor that requires the arguments `commonName` and `latinName`
- Create two new instances of the Bird class
  - `commonName` = Robin
  - `latinName` = Turdus migratorius
  - `commonName` = Eastern Towhee
  - `latinName` = Pipilo erythrophthalmus

Output

```
Common name: American Robin
Latin name: Turdus migratorius
________________________________
Common name: Eastern Towhee
Latin name: Pipilo erythrophthalmus
```

Array of Arguments

This exercise is based off chapter 6.2, *Contstructor Arguments*

A more flexible and readable way to create a constructor is using an array.

Build on what you created in the previous exercise.

Code

- Create a new file named `constructor-arguments.php` in your `asgn04-constructors` folder.
- Create a class named `Bird` with two public properties
  - `commonName`
  - `latinName`
- Create a constructor that uses an array called `args[]`.

```
public function __construct($args[]) {
}
```

- Use the null coalescing operator (shorthand ternary operator).
- Create two new instances of the Bird class
  - `commonName` = Acadian Flycatcher
  - `latinName` = Turdus migratorius
  - `commonName` = Eastern Towhee
  - `latinName` = Pipilo erythrophthalmus

Output

```
Common name: Acadian Flycatcher
Latin name: Empidonax virescens

Common name: Carolina Wren
Latin name: Thryothorus ludovicianus
```

Git

- Stage and commit your `asgn04-constructor`

Autoload

This exercise is based off chapter 6.7, *Undefined Classes*

The purpose of autoload is to only load classes you use from a class library. This saves on memory if you have a large library of classes.

- Create a new file named `autoload` in your `asgn04-constructors` folder.
- Create a new folder called `classes` in your `asgn04-constructors` folder.
- Create a file named `bird.class.php` inside your `classes` folder.

bird.class.php

Add the following code to your `bird.class.php` file

```php
Class Bird {

    public $commonName;
    public $latinName;
    public function __construct($args=[]) {
        $this->commonName = $args['commonName'] ?? NULL;
        $this->latinName = $args['latinName'] ?? NULL;
    }

}
```

Follow along with the video to create your `autoload.php` file.

It is easy to make mistakes when writing regular expressions, so here is the function and the registration.

```
function my_autoload($class) {
    if(preg_match('/\A\w+\Z/', $class)) {
        include 'classes/' . $class . '.class.php';
    }
}

spl_autoload_register('my_autoload');
```

You are required to create a new instance of a Acadian Flycatcher and echo its `commonName` .

Output

Acadian Flycatcher

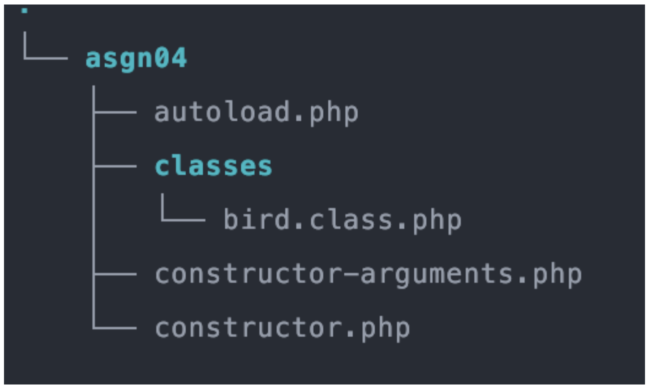
Git

Once your code is working correctly

- Stage and commit the `asgn04-constructor` branch.
- Switch to `main.
- Merge the `asgn04-constructor` branch.
- Push your code to your GitHub account. Use `git push --all` to include all of your branches.

Folder structure

When you are finished adding all of the files and folders, your folder tree should look like.



```
.
└── asgn04
    ├── autoload.php
    ├── classes
    │   └── bird.class.php
    ├── constructor-arguments.php
    └── constructor.php
```

Submit Work

Paste your GitHub URL into the comments section of Moodle.