Intro to Object Oriented Programming with PHP

a basic look into object-oriented programming

What we will cover

- Basic Terminology (Anatomy and usage of objects)
- Magic (Methods and Constants)
- Static Members (Properties and Methods)
- Getting Organized (Autoloading)
- Type Declarations

What is OOP?

- Object-Oriented Programing
- A programming concept that treats functions and data as objects.
- A programming methodology based on objects, instead of functions and procedures

OOP vs Procedural or Functional

OOP is built around the "nouns", the things in the system, and what they are capable of

Whereas procedural or functional programming is built around the "verbs" of the system, the things you want the system to do

Terminology

the single most important part

Anatomy of a Class

properties

methods

“this”

Object

Instance

Getters and Setters

Class

A template/blueprint that facilitates creation of objects. A set of program statements to do a certain task. Usually represents a noun, such as a person, place or thing.

Includes properties and methods — which are class functions

Object

- Instance of a class.
- In the real world object is a material thing that can be seen and touched.
- In OOP, object is a self-contained entity that consists of both data and procedures.

Instance

Single occurrence/copy of an object

There might be one or several objects, but an instance is a specific copy, to which you can have a reference

Example: Creating a Class and Instance

```php
class User { // class
  public $name; // property
  public function getSalutation() { // method
     return "Hello " . $this->name; // current object property
  }
}
$user1 = new User(); // first instance of object
$user1->name = "ada lovelace";
$user2 = new User(); // second instance of object
$user2->name = "GRACE HOPPER";
```

Example: Using the Object Instance

```
echo $user1->getSalutation();
echo "<br />\n";
echo $user2->getSalutation();
```

When the script is run, it will return:

```
Hello ada lovelace
Hello GRACE HOPPER
```

Challenge

Write a class with properties and methods

Instantiate and use an instance of that class

Getters and Setters

- Control how properties are accessed and updated
- Improved interoperability (reuse and testing)
- Allowing overriding control in children
- Allow different access levels (scope)

```
class User {
  public $name;
  public $title = "Mx.";
  public $acceptedTitles = ["Mr.", "Ms.", "Mrs.", "Mx."];
  public function getSalutation() {
    return "Hello " . $this->title . " " . $this->name;
  }
  public function setName($name) {
    $this->name = ucwords(strtolower($name));
  }
  public function setTitle($title) {
    $formatedTitle = trim(ucwords(strtolower($title)),".") . ".";
    if (in_array($formatedTitle, $this->acceptedTitles)) {
      $this->title = $formatedTitle;
    }
  }
}
```

Example: Using the Object Instance

```
$user = new User();
echo $user->getSalutation();
echo "<br />";
$user->setName("margaret hamilton");
echo $user->getSalutation();
echo "<br />";
$user->setTitle("ms.");
echo $user->getSalutation();
```

Example: Output

```
Hello Mx.
Hello Mx. Margaret Hamilton
Hello Ms. Margaret Hamilton
```

Challenge

Add getters and setters to control the formatting

Add additional properties and/or methods

Let's Add Some "Magic"

Magic Methods and Magic Constants

Magic Methods

- Setup just like any other method
- The Magic comes because they are **triggered** and not called

  __construct(), __destruct(), __call(), __callStatic(), __get(), __set(), __isset(), __unset(), __sleep(), __wakeup(), __toString(), __invoke(), __set_state(), __clone() and __debugInfo()
- For more see http://php.net/manual/en/language.oop5.magic.php

Magic Constants

- Constants that change depending on where they are used

  __LINE__, __FILE__, __DIR__, __FUNCTION__, __CLASS__, __TRAIT__, __METHOD__, __NAMESPACE__
- For more see http://php.net/manual/en/language.constants.predefined.php

Example: Constructor Method & Magic Methods

```
class User {
...
  public function __construct(string $name, string $title = '') {
     $this->setName($name);
     $this->setTitle($title);
  }
  public function __toString() : string {
     return $this->getSalutation() . ", " . __CLASS__;
  }
...
}
```

Example: Using the Object Instance

```
$user = new User("rasmus lerdorf", "mr");
echo $user;
```

When the script is run, it will return:

```
Hello Mr. Rasmus Lerdorf, User
```

Challenge

Add some Magic Methods and Constants

For Method http://php.net/manual/en/language.oop5.magic.php

For Constants http://php.net/manual/en/language.constants.predefined.php

Static Members

properties and methods without an object

Defining Static

Static: "lacking in movement, action, or change"

- ▷ Properties and Methods can be defined as static
- ▷ Are accessed without the creation of an object
- ▷ Can be used to group constants and functions
- ▷ Can be autoloaded
- ▷ Does not save state

Static Properties and Methods

- Define a static members, use the keyword static
- Access using the operator(::)
- Class may contain both static and non static members
- Static methods **cannot** access non-static variables
- Non-static methods **can** access static method

```
class User {
...
  public static $encouragements = [
      "You are beautiful!",
      "You have this!",
      "Stop touching your face!"
  ];
  public static function encourage() {
    $int = array_rand(self::$encouragements, 1);
    return self::$encouragements[$int];
  }
  public function __toString() : string {
    return $this->getSalutation() . ", " . __CLASS__
    . ". " . self::encourage();
  }
}
```

Example: Using the Object Instance

```
echo User::encourage();
echo "<br />\n";
echo User::$encouragements[0];
echo "<br />\n";
$user = new User("rasmus lerdorf", "mr");
echo $user;
```

When the script is run, it will return:

```
You have this!
You are beautiful!
Hello Mr. Rasmus Lerdorf User. Stop touching your face!
```

Challenge

Define a static property

Define a static method

Call a static property from static method

Call a static property or method outside the class

Call a static property or method from a non-static method

Getting Organized

Autoloading Classes

Manually Loading Classes

```
<?php
require "classes/User.php";

$user = new User("rasmus lerdorf", "mr");
echo $user;
```

Autoloading Classes

```php
<?php
spl_autoload_register(function ($class) {
  $file = 'classes/' . $class . '.php';
  if (file_exists($file)) {
    require $file;
  }
});

$user = new User("rasmus lerdorf", "mr");
echo $user;
```

Challenge

Move your class into its own file

Add an autoloader

Type Declarations

Parameter Types and Return Types

Types in PHP

Specified

- boolean (bool)
- integer (int)
- float
- string
- array
- object (User)

Nullable (7.1)

- ?bool
- ?int
- ?float
- ?string
- ?array
- ?object

For more information https://www.php.net/manual/en/language.types.intro.php

How PHP Uses Types

- By default PHP 7 remains a weakly typed language
- Uses type juggling to make things work
  - One type will be cast to a value of another type
- Developers can turn on strict types
  - declare(strict_types=1);

```
class User {
  public $name;
  public $title = "Mx.";
  public $acceptedTitles = ["Mr.", "Ms.", "Mrs.", "Mx."];
  public function getSalutation() : string {
    return "Hello " . $this->title . " " . $this->name;
  }
  public function setName(string $name) {
    $this->name = ucwords(strtolower($name));
  }
  public function setTitle(string $title) {
    $formatedTitle = trim(ucwords(strtolower($title)),".") . ".";
    if (in_array($formatedTitle, $this->acceptedTitles)) {
      $this->title = $formatedTitle;
    }
  }
}
```

```php
class User {
  public string $name;
  public string $title = "Mx.";
  public array $acceptedTitles = ["Mr.", "Ms.", "Mrs.", "Mx."];
  public function getSalutation() : string {
    return "Hello " . $this->title . " " . $this->name;
  }
  public function setName(string $name) {
    $this->name = ucwords(strtolower($name));
  }
  public function setTitle(string $title) {
    $formatedTitle = trim(ucwords(strtolower($title)),".") . ".";
    if (in_array($formatedTitle, $this->acceptedTitles)) {
      $this->title = $formatedTitle;
    }
  }
}
```

Challenge

Add type declarations to your methods

Try with and without setting declare(strict_types=1);

Experiment with strict class file vs strict calling file

Questions and Answers

Thank you for your participation