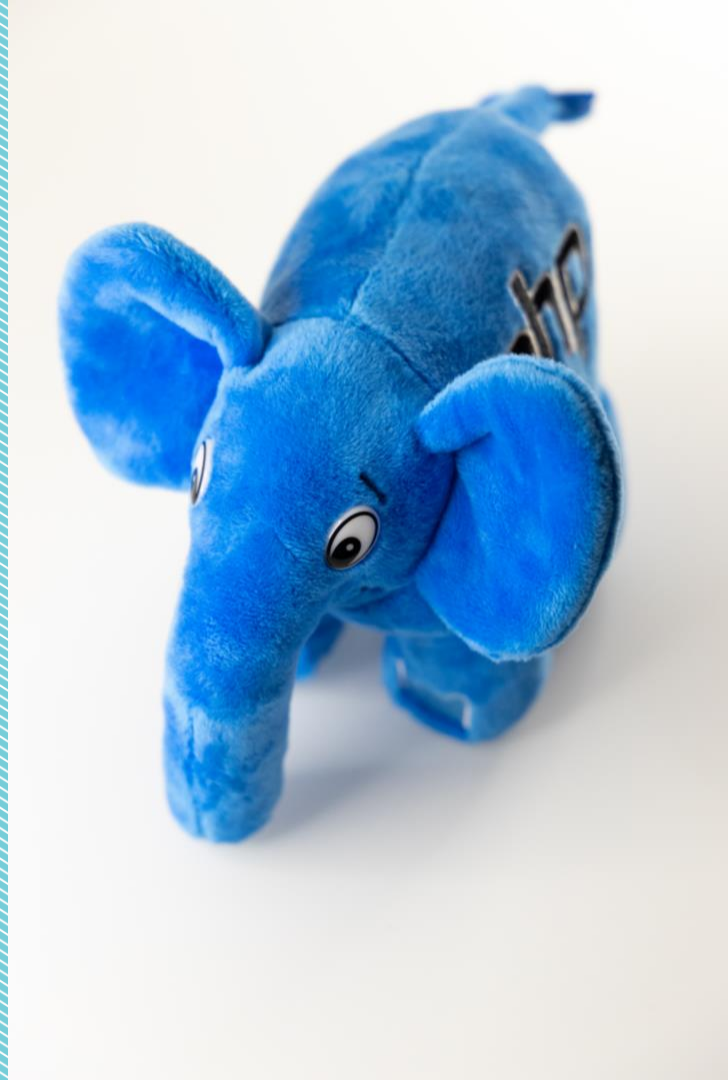
OOP Basics

Disclaimer: вы смотрите просто запись лекции,
это **НЕ** специально подготовленный видеокурс!



Disclaimer!

This course part is not about “OOP in general”, it is about “OOP **in PHP**”.

That is why we’ll not discuss abstracts paradigms and concepts, we’ll concentrate on syntax, PHP specifics, some best practices, etc.

Main OOP Principles: Abstraction

Abstraction is the concept of OOP that enforces handling implementation complexity by hiding unnecessary details from end-users.

```php
<?php

class Math
{
    public static function getMedian(array $inputNumbersArray): float|int
    {
        sort($inputNumbersArray, SORT_NUMERIC);
        $inputNumbersArrayCount = count($inputNumbersArray);
        $centralElementIndex = floor($inputNumbersArrayCount / 2);
        if ($inputNumbersArrayCount & 1) {
            return $inputNumbersArray[$centralElementIndex];
        } else {
            return ($inputNumbersArray[$centralElementIndex - 1] +
                    $inputNumbersArray[$centralElementIndex]) / 2;
        }
    }
}
```

Complex implementation.

Simple usage.

```php
$testArrayOne = [3, 1, 8];
echo "MedianOne = " . Math::getMedian($testArrayOne) . "\n";
// MedianOne = 3

$testArrayTwo = [3, 1, 8, 4];
echo "MedianTwo = " . Math::getMedian($testArrayTwo) . "\n";
// MedianTwo = 3.5
```

Main OOP Principles: Encapsulation

Encapsulation is the concept of OOP that enforces binding the data and data manipulation, and protection of both from outside interference.

```php
<?php

class URL
{
    public const STATUS_NOT_INITIALIZED = 0b0000;
    public const STATUS_URI_SET = 0b0001;
    public const STATUS_REQUEST_COMPLETED = 0b0010;

    private $currentStatus = self::STATUS_NOT_INITIALIZED;
    private $currentURI = '';

    public function setURI(string $URI): bool
    {
        if (filter_var($URI, FILTER_VALIDATE_URL)) {
            $this->currentURI = $URI;
            $this->currentStatus = $this->currentStatus | self::STATUS_URI_SET;
            return true;
        } else {
            return false;
        }
    }

    public function getStatus() : int {
        return $this->currentStatus;
    }
}
```

It is impossible to mess with **currentStatus** and/or **currentURI** data.

We can only use pre-defined mechanisms and observe effects.

```php
$url = new URL;
$url->setURI('https://google.com');
echo "URL status: " . $url->getStatus();
// URL status: 1
```

Main OOP Principles: Inheritance

Inheritance is the concept of OOP that allows basing a class upon another class retaining similar implementation.

```php
<?php

class StringsManipulator {
    protected $currentString;

    public function __construct(string $initialString) {
        $this->currentString = $initialString;
    }

    public function getLengthInBytes() : int {
        return strlen($this->currentString);
    }

    public function getLengthInSymbols() : int {
        return strlen($this->currentString);
    }
}

class UTF8StringsManipulator extends StringsManipulator {

    public function getLengthInSymbols() : int {
        return mb_strlen($this->currentString, 'UTF8');
    }
}
```

Parent class contains the general code.

Child class inherits most of that code and re-defines (overrides) some behavior.

```php
$singleByteString = new StringsManipulator('Test');
echo 'singleByteString bytes = ' .
     $singleByteString->getLengthInBytes() . "\n"; // 4
echo 'singleByteString symbols = ' .
     $singleByteString->getLengthInSymbols() . "\n"; // 4

$UTF8String = new UTF8StringsManipulator('Тест');
echo 'UTF8String bytes = ' .
     $UTF8String->getLengthInBytes() . "\n"; // 8
echo 'UTF8String symbols = ' .
     $UTF8String->getLengthInSymbols() . "\n"; // 4
```

Main OOP Principles: Polymorphism

Polymorphism is the concept of OOP that allows provision of a single interface to entities of different types.

```php
<?php

abstract class Connection {
    private $uniqueId;

    public function __construct(){
        $this->uniqueId = uniqid();
    }

    public function getUniqueId(): string
    {
        return $this->uniqueId;
    }
}

class HttpConnection extends Connection {
    public function connect() {}
}

class HttpsConnection extends Connection {
    public function connect() {}
}

class FtpConnection extends Connection {
    public function connect() {}
}
```

These are three different connection types. Their **connect** methods have different implementations…

… and yet – we don't care: as long as we have any connection, we may easily call **connect** method.

```php
class ConnectionManager{
    private $connectionsPool = [];

    public function addActiveConnection(Connection $connection) {
        $this->connectionsPool[$connection->getUniqueId()] = $connection;
        $this->connectionsPool[$connection->getUniqueId()]->connect();
    }
}
```

Further learning…

https://www.php.net/manual/en/language.oop5.php

Learn Java, it really helps with PHP OOP understanding.

Seriously. Todays' OOP in PHP is mostly 'Java OOP'.

OOP Basics

Disclaimer: вы смотрите просто запись лекции,
это **НЕ** специально подготовленный видеокурс!