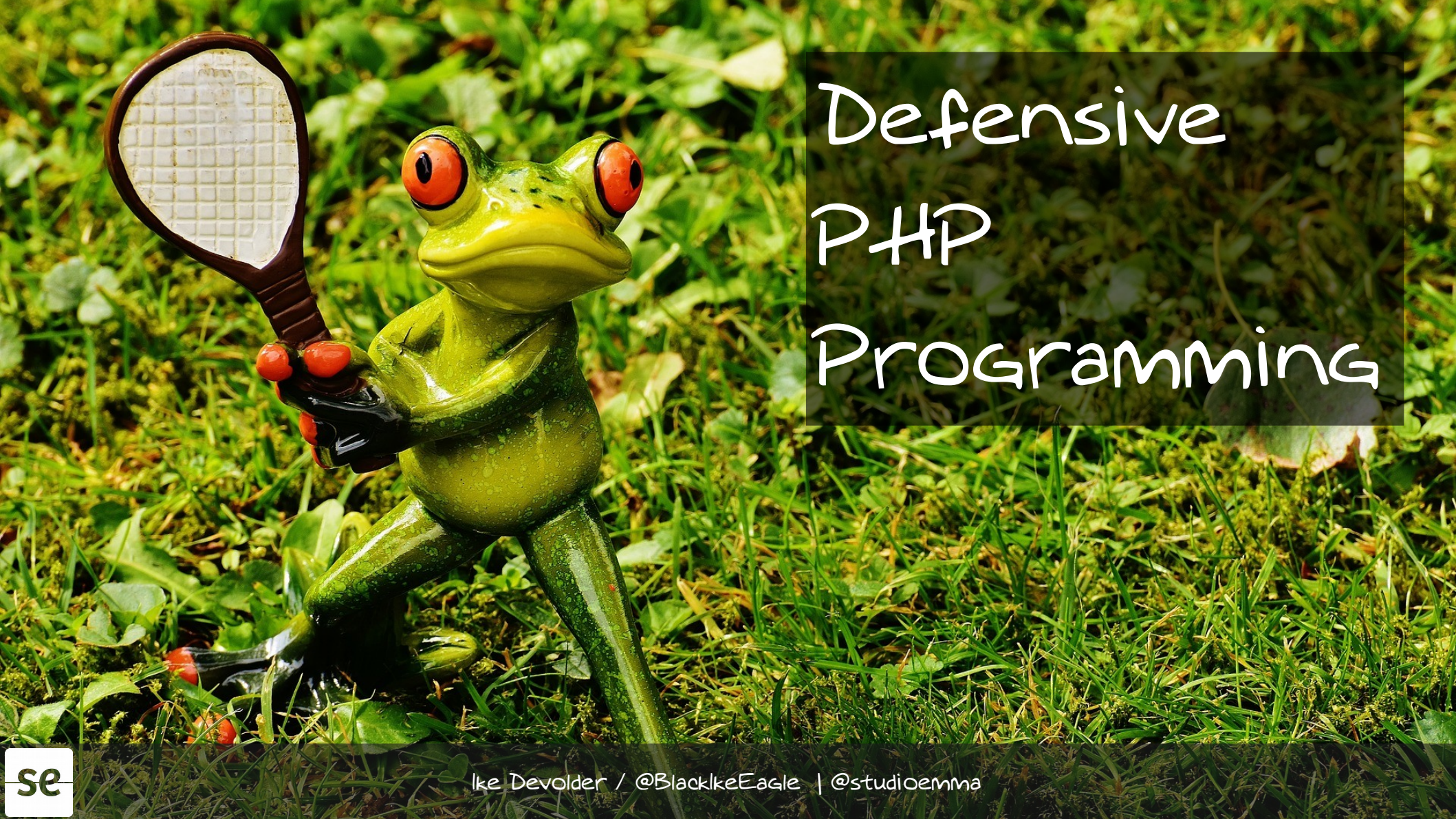


A green tree frog with large red eyes is sitting on a grassy surface, holding a tennis racket in its front paws. The frog is looking towards the camera. The background is a dense field of green grass and small plants.

Defensive PHP Programming

Disclaimer





Hi,

I'm Ike,

@BlackIkeEagle on Twitter

Developer / Operations
at Studio Emma



studio emma

internet expertise



Magento[®]

Silver solution partner

PIMCORE[®]

Gold solution partner

Object Calisthenics



KATAs



#1 One indentation level
per method



```
<?php
```

```
class IndentationLevel
{
    public function doSomething($array, $string)
    {
        foreach ($array as $key => &$value) {
            if (stripos($key, 'selector')) {
                if (!empty($value)) {
                    $value .= $string;
                } else {
                    $value = $string;
                }
            } else {
                if (!empty($value)) {
                    $value = $string . $value;
                } else {
                    $value = $string;
                }
            }
        }
        return $array;
    }
}
```

```
<?php
```

```
class IndentationLevel
{
    public function doSomething($array, $string)
    {
        foreach ($array as $key => &$value) {
            $value = $this->updateValueByKey(
                $key,
                $value
            );
        }
        return $array;
    }

    private function updateValueByKey($key, $value, $string)
    {
        if (stripos($key, 'selector')) {
            return $this->updateSelectorValue($value, $string);
        } else {
            return $this->updateOtherValue($value, $string);
        }
    }

    private function updateSelectorValue($value, $string)
    {
        if (!empty($value)) {
            $value .= $string;
        } else {
            $value = $string;
        }
        return $value;
    }

    private function updateOtherValue($value, $string)
    {
        if (!empty($value)) {
            $value = $string . $value;
        } else {
            $value = $string;
        }
        return $value;
    }
}
```



#2 Avoid else



```
<?php
```

```
class AvoidElse
```

```
{
```

```
    public function doSomething($array, $string)
```

```
    {
```

```
        if (empty($array)) {
```

```
            $array[] = $string;
```

```
        } else {
```

```
            $array = [$string];
```

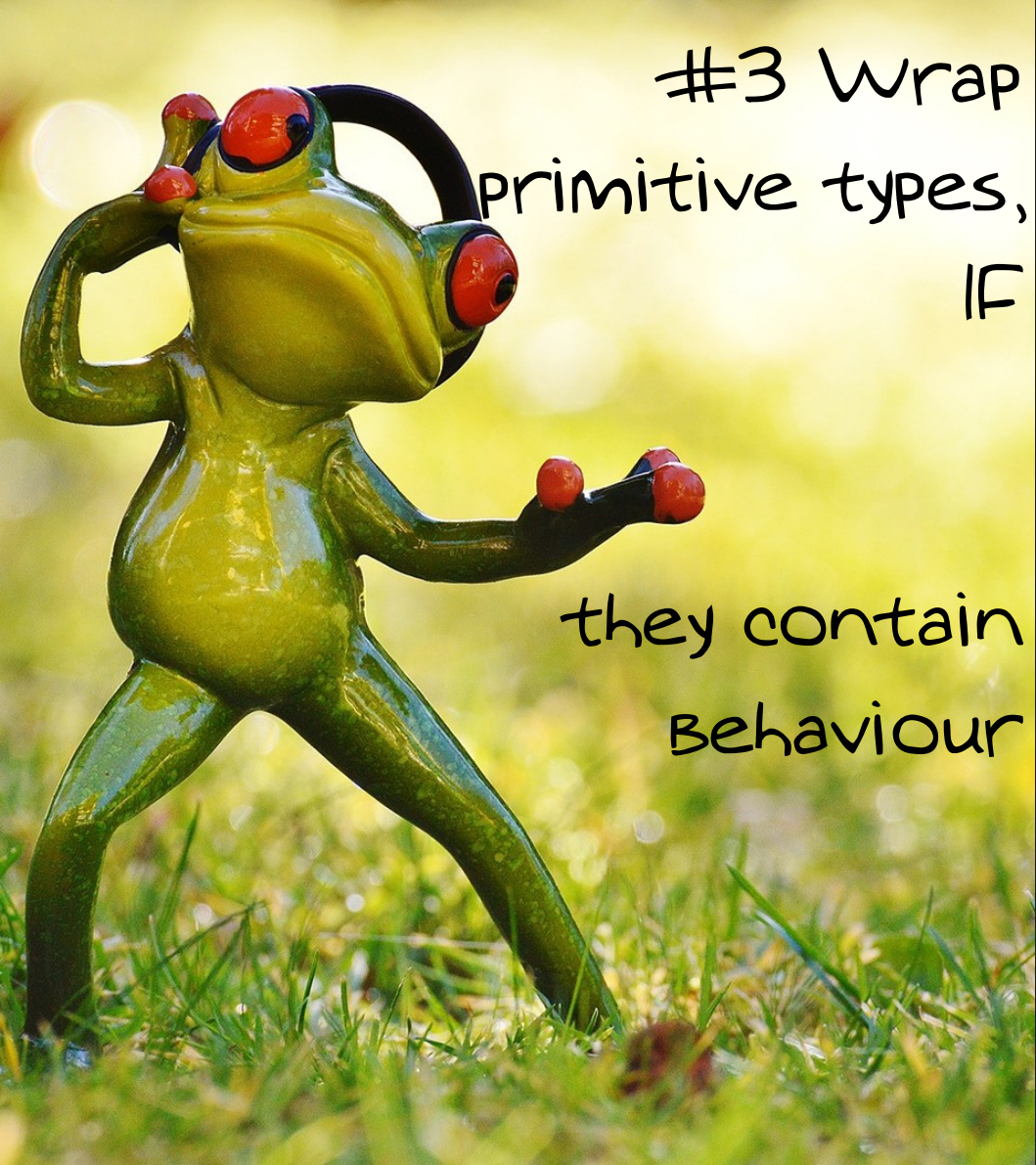
```
        }
```

```
    }
```

```
}
```

```
<?php  
  
class AvoidElse  
{  
    public function doSomething($array, $string)  
    {  
        if (empty($array)) {  
            $array = [];  
        }  
        $array[] = $string;  
    }  
}
```





#3 Wrap
primitive types,
IF

they contain
Behaviour

```
<?php  
  
class PrimitiveTypesBehaviour  
{  
    public function doSomething()  
    {  
        $component->repaint(true);  
    }  
}
```

```
<?php  
  
class PrimitiveTypesBehaviour  
{  
    public function doSomething()  
    {  
        $component->repaint( new Animate(true) );  
    }  
}
```



#4 only one

->

per line



```
<?php
```

```
class Abbreviation  
{
```

```
    private $mx = 1000;  
    private $my = 1000;
```

```
    public function image($sx, $sy)  
    {
```

```
        $ox = $sx;  
        $oy = $sy;
```

```
        if ($sx > $this->mx) {  
            $rt = $this->mx / $sx;  
            $ox = $this->mx;  
            $oy = $sy * $rt;  
        } elseif ($sy > $this->my) {  
            $rt = $this->my / $sy;  
            $oy = $this->my;  
            $ox = $ox * $rt;  
        }  
    }
```

```
    return [$ox, $oy];
```

```
}
```

```
}
```

#5 No aBBreviation





```
<?php
```

```
class Abbreviation
```

```
{
```

```
    private $maxWidth = 1000;
```

```
    private $maxHeight = 1000;
```

```
    public function image($sourceWidth, $sourceHeight)
    {
```

```
        $thumbnailWidth = $sourceWidth;
```

```
        $thumbnailHeight = $sourceHeight;
```

```
        if ($sourceWidth > $this->maxWidth) {
```

```
            $factor = $this->maxWidth / $sourceWidth;
```

```
            $thumbnailWidth = $this->maxWidth;
```

```
            $thumbnailHeight = $sourceHeight * $factor;
```

```
        } elseif ($sourceHeight > $this->maxHeight) {
```

```
            $factor = $this->maxHeight / $sourceHeight;
```

```
            $thumbnailHeight = $this->maxHeight;
```

```
            $thumbnailWidth = $sourceWidth * $factor;
```

```
        }
```

```
        return [$thumbnailWidth, $thumbnailHeight];
```

```
    }
```

```
}
```



#6 Keep your classes small

#1 Limit your instance variables



```
<?php
```

```
class Customer
{
    public function __construct(
        $firstname,
        $lastname,
        $email,
        $street,
        $number,
        $postalcode,
        $city,
        $country
        // ...
    ) {
        // ... some code
    }
}
```

```
<?php
```

```
class Customer
{
    public function __construct(
        Name $name,
        Email $email,
        Address $address
        // ...
    ) {
        // ... some code
    }
}
```



```
<?php  
  
class Game  
{  
    private $score = 0;  
  
    public function getScore()  
    {  
        return $this->score;  
    }  
  
    public function setScore($score)  
    {  
        $this->score = $score;  
    }  
}  
  
$game = new Game();  
$game->setScore($game->getScore() + 1);
```



#8 Don't use
getters and
setters



```
<?php
```

```
class Game  
{
```

```
    private $score = 0;
```

```
    public function currentScore()  
    {  
        return $this->score;  
    }
```

```
    public function goal()  
    {  
        $this->score += 1;  
    }
```

```
}
```

```
$game = new Game();
```

```
$game->goal();
```

#9 Document
your code







Techniques
for safer
code

Fail early and
avoid deep nesting



```
<?php
```

```
class FailEarly
```

```
{
```

```
    public function charsRemover(
```

```
        string $inputText,
```

```
        array $charsToRemove,
```

```
        int $length) :string
```

```
    {
```

```
        if (strlen($inputText) > 0) {
```

```
            if (count($charsToRemove) > 0) {
```

```
                if ($length > 0) {
```

```
                    return str_replace(
```

```
                        $charsToRemove,
```

```
                        '',
```

```
                        $inputText,
```

```
                        $length
```

```
                    );
```

```
                }
```

```
            }
```

```
        }
```

```
        return $inputText;
```

```
    }
```

```
}
```

```

<?php

class FailEarly
{
    public function charsRemover(
        string $inputText,
        array $charsToRemove,
        int $length) :string
    {
        if (strlen($inputText) > 0) {
            if (count($charsToRemove) > 0) {
                if ($length > 0) {
                    return str_replace(
                        $charsToRemove,
                        '',
                        $inputText,
                        $length
                    );
                }
            }
        }

        return $inputText;
    }
}

```

```

<?php

class FailEarly
{
    public function charsRemover(
        string $inputText,
        array $charsToRemove,
        int $length) :string
    {
        if (strlen($inputText) === 0) {
            return $inputText;
        }

        if (count($charsToRemove) === 0) {
            return $inputText;
        }

        if ($length <= 0) {
            return $inputText;
        }

        return str_replace(
            $charsToRemove,
            '',
            $inputText,
            $length
        );
    }
}

```

Don't have unexpected code paths



```
<?php
```

```
class NoUnexpectedBehaviour
{
    public function changeStatus(
        Order $order,
        string $status) :void
    {
        switch ($status) {
            case 'placed':
                $order->confirm();
                break;
            case 'paymentRecieved':
                $order->paymentReceived();
                break;
            case 'shipped':
                $order->shipped();
                break;
        }
    }
}
```

```
<?php
```

```
class NoUnexpectedBehaviour
{
    public function changeStatus(
        Order $order,
        string $status) :void
    {
        switch ($status) {
            case 'placed':
                $order->confirm();
                break;
            case 'paymentRecieved':
                $order->paymentReceived();
                break;
            case 'shipped':
                $order->shipped();
                break;
        }
    }
}
```

```
<?php
```

```
class NoUnexpectedBehaviour
{
    public function changeStatus(
        Order $order,
        string $status) :void
    {
        switch ($status) {
            case 'placed':
                $order->confirm();
                break;
            case 'paymentRecieved':
                $order->paymentReceived();
                break;
            case 'shipped':
                $order->shipped();
                break;
            default:
                throw new UnknownOrderStatusException();
        }
    }
}
```

Reverse conditions



```
<?php
```

```
class ReverseConditions
```

```
{
```

```
    public function condition(
```

```
        string $status,
```

```
        bool $readOnly,
```

```
        int $id) :bool
```

```
    {
```

```
        if ($status === 'invalid') {
```

```
            return false;
```

```
        }
```

```
        if ($readOnly === false) {
```

```
            return false;
```

```
        }
```

```
        if ($id >= 123) {
```

```
            return false;
```

```
        }
```

```
        if ($id === 122) {
```

```
            return false;
```

```
        }
```

```
        return true;
```

```
    }
```

```
}
```

```
<?php

class ReverseConditions
{
    public function condition(
        string $status,
        bool $readOnly,
        int $id) :bool
    {
        if ($status === 'invalid') {
            return false;
        }

        if ($readOnly === false) {
            return false;
        }

        if ($id >= 123) {
            return false;
        }

        if ($id === 122) {
            return false;
        }

        return true;
    }
}
```

```
<?php

class ReverseConditions
{
    public function condition(
        string $status,
        bool $readOnly,
        int $id) :bool
    {
        if ('invalid' === $status) {
            return false;
        }

        if (false === $readOnly) {
            return false;
        }

        if (123 <= $id) {
            return false;
        }

        if (122 === $id) {
            return false;
        }

        return true;
    }
}
```

```
<?php

class ReverseConditions
{
    public function condition(
        string $status,
        bool $readOnly,
        int $id) :bool
    {
        if ('invalid' === $status) {
            return false;
        }

        if (false === $readOnly) {
            return false;
        }

        if (123 <= $id) {
            return false;
        }

        if (122 === $id) {
            return false;
        }

        return true;
    }
}
```

```
<?php

class ReverseConditions
{
    public function condition(
        string $status,
        bool $readOnly,
        int $id) :bool
    {
        if ('invalid' === $status) {
            return false;
        }

        if (false === $readOnly) {
            return false;
        }

        if ($id >= 123) {
            return false;
        }

        if (122 === $id) {
            return false;
        }

        return true;
    }
}
```

New functionality == new method



```
<?php
```

```
class AvoidControlParameters
```

```
{
```

```
    public function login(
```

```
        string $username,
```

```
        string $password,
```

```
        bool $freshlyRegistered) :bool
```

```
    {
```

```
        // ... some code
```

```
    }
```

```
}
```

```
<?php
```

```
class AvoidControlParameters
{
    public function login(
        string $username,
        string $password,
        bool $freshlyRegistered) :bool
    {
        // ... some code
    }
}
```

```
<?php
```

```
class AvoidControlParameters
{
    public function login(
        string $username,
        string $password) :bool
    {
        // ... some code
    }

    public function loginAsFreshlyRegistered(
        string $username,
        string $password) :bool
    {
        // ... some code
    }
}
```



```
<?php
```

```
class AvoidParameterCreep
```

```
{
```

```
    public function getListForUser(
```

```
        int $userId,
```

```
        string $type = 'default')
```

```
    {
```

```
        $query = "SELECT * FROM `test`";
```

```
        switch ($type) {
```

```
        case 'freshTokens':
```

```
            // some magic additions to $query
```

```
            break;
```

```
        case 'invalidTokens':
```

```
            // some other magic additions to $query
```

```
            break;
```

```
        default:
```

```
            // keep original behaviour
```

```
            break;
```

```
        }
```

```
        // some extra code
```

```
        return $result;
```

```
    }
```

```
}
```

```
<?php
```

```
class AvoidParameterCreep
{
    public function getListForUser(
        int $userId,
        string $type = 'default')
    {
        $query = "SELECT * FROM `test`";

        switch ($type) {
            case 'freshTokens':
                // some magic additions to $query
                break;
            case 'invalidTokens':
                // some other magic additions to $query
                break;
            default:
                // keep original behaviour
                break;
        }

        // some extra code

        return $result;
    }
}
```

```
<?php
```

```
class AvoidParameterCreep
{
    private function getFreshTokensForUser(
        int $userId)
    {
    }

    private function getInvalidTokensForUser(
        int $userId)
    {
    }

    private function getUsableTokensForUser(
        int $userId)
    {
    }

    public function getListForUser(
        int $userId,
        string $type = 'default')
    {
        switch ($type) {
            case 'freshTokens':
                return $this->getFreshTokensForUser(
                    $userId
                );
                break;
            case 'invalidTokens':
                return $this->getInvalidTokensForUser(
                    $userId
                );
                break;
            default:
                return $this->getUsableTokensForUser(
                    $userId
                );
        }
    }
}
```

```
<?php
```

```
class AvoidParameterCreep
```

```
{
```

```
    public function getListForUser(
```

```
        int $userId,
```

```
        string $type = 'default')
```

```
    {
```

```
        $query = "SELECT * FROM `test`";
```

```
        switch ($type) {
```

```
            case 'freshTokens':
```

```
                // some magic additions to $query
```

```
                break;
```

```
            case 'invalidTokens':
```

```
                // some other magic additions to $query
```

```
                break;
```

```
            default:
```

```
                // keep original behaviour
```

```
                break;
```

```
        }
```

```
        // some extra code
```

```
        return $result;
```

```
    }
```

```
}
```

```
<?php
```

```
class AvoidParameterCreep
```

```
{
```

```
    public function getFreshTokensForUser(
```

```
        int $userId)
```

```
    {
```

```
    }
```

```
    public function getInvalidTokensForUser(
```

```
        int $userId)
```

```
    {
```

```
    }
```

```
    public function getUsableTokensForUser(
```

```
        int $userId)
```

```
    {
```

```
    }
```

```
}
```



Avoid optional
dependencies

```
<?php
```

```
class AvoidOptionalDependencies
```

```
{
```

```
    const GUEST = 1;
```

```
    const VISITOR = 2;
```

```
    const ADMIN = 3;
```

```
    private $user = null;
```

```
    private $logger = null;
```

```
    public function __construct(
```

```
        UserInterface $user,
```

```
        Logger $logger = null)
```

```
    {
```

```
        // initialize
```

```
    }
```

```
    public function checkAcl()
```

```
    {
```

```
        if (self::GUEST === $user->acl()) {
```

```
            if (! empty($this->logger)) {
```

```
                $this->logger->log(/*...*/);
```

```
            }
```

```
        }
```

```
        // some other code
```

```
    }
```

```
}
```

```
<?php
```

```
class AvoidOptionalDependencies
```

```
{
```

```
    const GUEST = 1;
```

```
    const VISITOR = 2;
```

```
    const ADMIN = 3;
```

```
    private $user = null;
```

```
    private $logger = null;
```

```
    public function __construct(
```

```
        UserInterface $user,
```

```
        Logger $logger = null)
```

```
    {
```

```
        // initialize
```

```
    }
```

```
    public function checkAcl()
```

```
    {
```

```
        if (self::GUEST === $user->acl()) {
```

```
            if (! empty($this->logger)) {
```

```
                $this->logger->log(/*...*/);
```

```
            }
```

```
        }
```

```
        // some other code
```

```
    }
```

```
}
```

```
<?php
```

```
class AvoidOptionalDependencies
```

```
{
```

```
    const GUEST = 1;
```

```
    const VISITOR = 2;
```

```
    const ADMIN = 3;
```

```
    private $user = null;
```

```
    private $logger = null;
```

```
    public function __construct(
```

```
        UserInterface $user,
```

```
        LoggerInterface $logger)
```

```
    {
```

```
        // initialize
```

```
    }
```

```
    public function checkAcl()
```

```
    {
```

```
        if (self::GUEST === $user->acl()) {
```

```
            $this->logger->log(/*...*/);
```

```
        }
```

```
        // some other code
```

```
    }
```

```
}
```



Avoid mixed return
types


```
<?php

class AvoidMixedReturnTypes
{
    public function frog($id)
    {
        $result = $this->repository('frog')
            ->id($id);

        if (empty($result)) {
            return false;
        }

        return new Frog($result->name());
    }
}
```

```
<?php

class AvoidMixedReturnTypes
{
    public function frog($id)
    {
        $result = $this->repository('frog')
            ->id($id);

        if (empty($result)) {
            return new NoFrog();
        }

        return new Frog($result->name());
    }
}
```

```
<?php
```

```
class AvoidMixedReturnTypes
{
    public function frog($id)
    {
        $result = $this->repository('frog')
            ->id($id);

        if (empty($result)) {
            return false;
        }

        return new Frog($result->name());
    }
}
```

```
<?php
```

```
class AvoidMixedReturnTypes
{
    public function frog($id)
    {
        $result = $this->repository('frog')
            ->id($id);

        if (empty($result)) {
            throw new NoFrogFoundException();
        }

        return new Frog($result->name());
    }
}
```

Value objects




```

<?php

class ValueObjects
{
    /**
     * @param mixed $status
     *   int 1 order placed
     *   int 2 payment received
     *   bool true order shipped
     *   null invalid order
     */
    public function changeStatus($status)
    {
        /* ... some code ... */
    }
}

```

```

<?php

class ValueObjects
{
    public function changeStatus(Status $status)
    {
        /* ... some code ... */
    }
}

class Status
{
    const ORDER_PLACED = 1;
    const PAYMENT_OK = 2;
    const SHIPPED = true;
    const INVALID = null;

    public function __construct($status)
    {
        $this->validateStatus($status);
        $this->status = $status;
    }

    private function validateStatus($status)
    {
        if (! $valid) {
            throw new \InvalidArgumentException(
                'Invalid status given'
            );
        }
    }
}

```



Immutability



```
<?php
```

```
class Immutable
{
    private $updateTime;

    public function updateTime(\DateTime $date)
    {
        $this->updateTime = $date;
    }
}
```

```
$now = new \DateTime('now');
$immutable = new Immutable();
$immutable->updateTime($now);
/* ... more code ... */
sleep(15);
$now->setTimestamp(time());
```

```
<?php

class Immutable
{
    private $updateTime;

    public function updateTime(\DateTime $date)
    {
        $this->updateTime = $date;
    }
}

$now = new \DateTime('now');
$immutable = new Immutable();
$immutable->updateTime($now);
/* ... more code ... */
sleep(15);
$now->setTimestamp(time());
```

```
<?php

class Immutable
{
    private $updateTime;

    public function updateTime(
        \DateTimeInterface $date)
    {
        $this->updateTime = $date;
    }
}

$now = new \DateTimeImmutable('now');
$immutable = new Immutable();
$immutable->updateTime($now);
/* ... more code ... */
sleep(15);
$now = $now->setTimestamp(time());
```

Interfaces ≠ Exceptions



Add logging



```

<?php

class ProcessService
{
    public function process()
    {
        if ($orderIsIncomplete) {
            throw new IncompleteOrderException();
        }

        if ($paymentIsPending) {
            throw new PaymentPendingException();
        }

        /* ... process things */
    }
}

```

```

<?php

class OrderProcessController
{
    public function processAction(
        $request,
        $response)
    {
        $input = $request->input();

        $processor = new ProcessService();

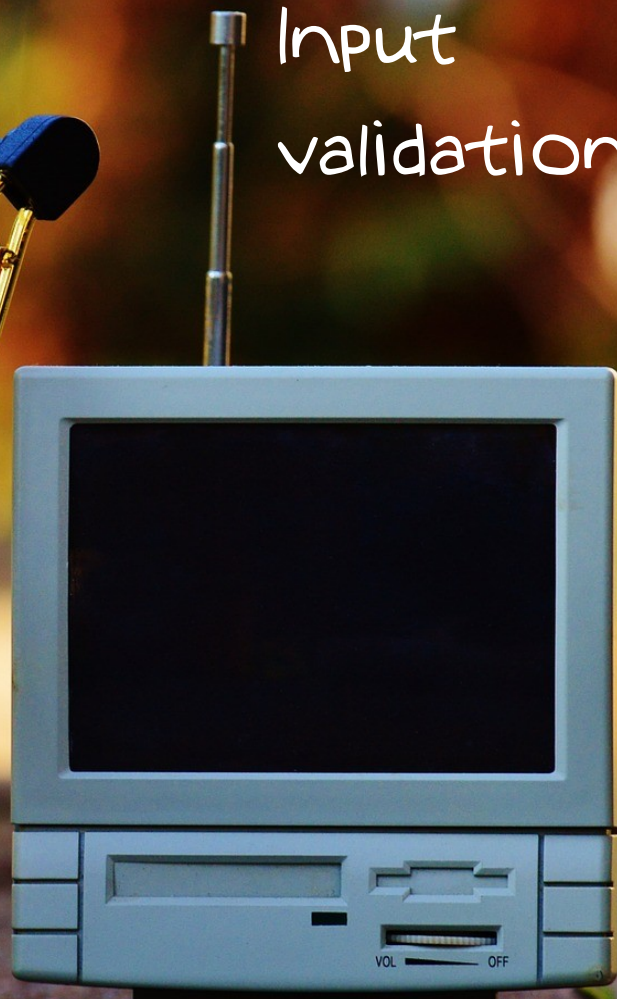
        try {
            $processor->process();
        } catch (PaymentPendingException $e) {
            \Logger::info(
                'payment is still pending',
                $context
            );
            $response->status(302)
                ->action('payment-pending');
        } catch (\Throwable $t) {
            \Logger::error(
                'something unexpected happend',
                $context
            );
            $response->status(500)
                ->action('error');
        }
        return $response;
    }
}

```

Threats



Input
validation



Prepared
statements



```
<?php
```

```
class PreparedStatements
```

```
{
```

```
    public function login($user, $pass)
```

```
    {
```

```
        $query = 'SELECT * FROM `user`'
```

```
        . ' WHERE'
```

```
        . " `user` = '" . $user . "'"
```

```
        . ' AND'
```

```
        . " `password` = '" . $pass . "'";
```

```
        $result = $this->db->query($query);
```

```
        /* ... further processing ... */
```

```
    }
```

```
}
```

```

<?php

class PreparedStatements
{
    public function login($user, $pass)
    {
        $query = 'SELECT * FROM `user`'
            . ' WHERE'
            . " `user` = '" . $user . "'"
            . ' AND'
            . " `password` = '" . $pass . "'";

        $result = $this->db->query($query);
        /* ... further processing ... */
    }
}

```

```

<?php

class PreparedStatements
{
    public function login($user, $pass)
    {
        $query = 'SELECT * FROM `user`'
            . ' WHERE'
            . ' `user` = ?'
            . ' AND'
            . ' `password` = ?';

        $statement = $this->db->prepare($query);
        $result = $statement->execute(
            [
                $user,
                $pass,
            ]
        );
        /* ... further processing ... */
    }
}

```

Output escape





Use tools

- PHP_CodeSniffer
- PHPCompatibility
- PHP-CS-Fixer
- Grumphp

Done



Ike Devolder / @BlackIkeEagle | @studioemma

Questions?



Thanks

<https://joind.in/talk/29847>

I'm Ike, Developer / Operations
at Studio Emma



@BlackIkeEagle



PHP-WVL



DockerWest



Parchive



BlackIkeEagle



Archlinux TU



Vdebug



herecura.eu

Frog Photos: https://pixabay.com/en/photos/user:alexas__fotos%20ofrog/

