



The PHP Company

# Leveraging Zend\_Form Decorators

Matthew Weier O'Phinney  
Project Lead  
Zend Framework

# What is a decorator?

# In Zend\_Form...

---

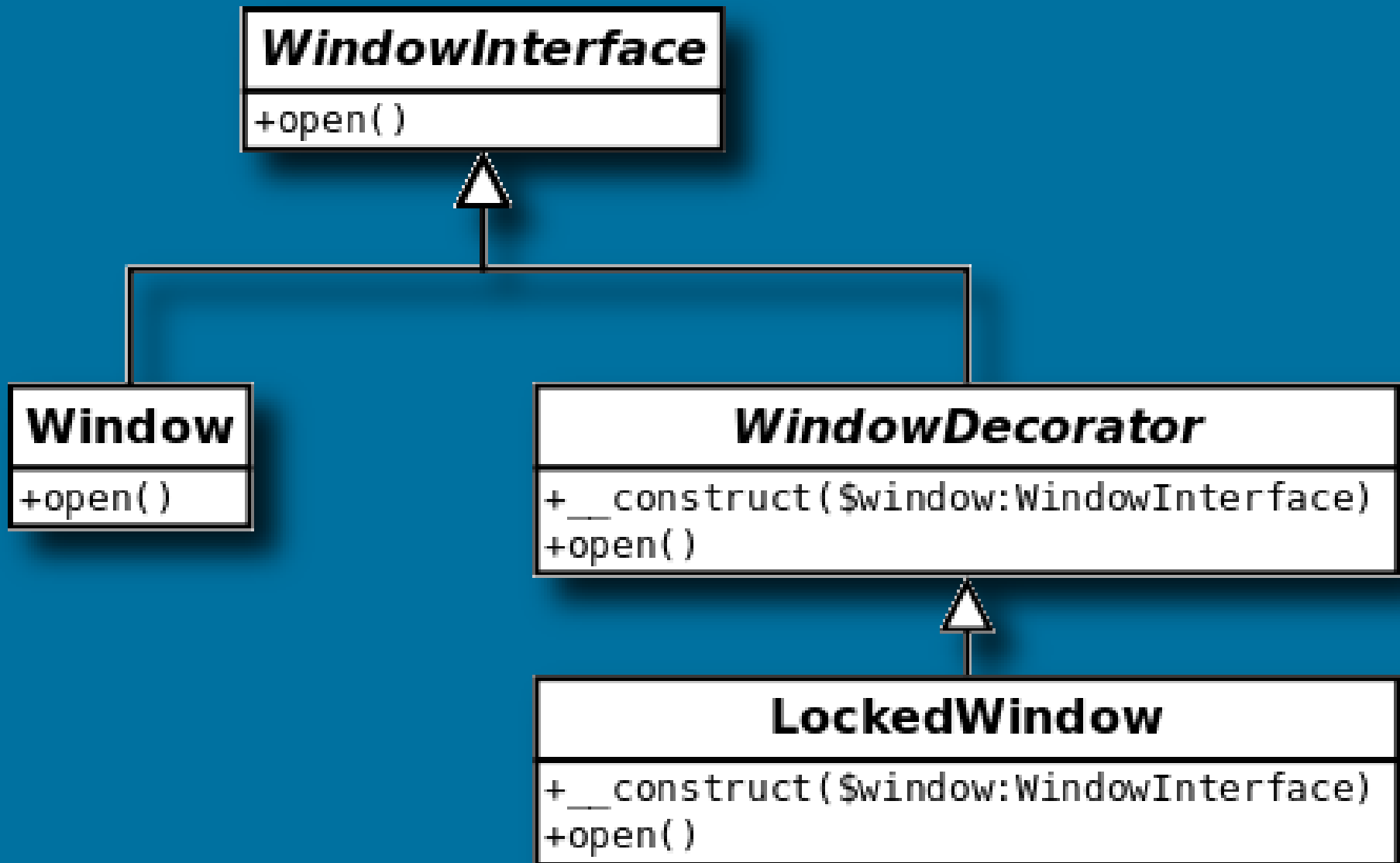
- Combination *Decorator* and *Strategy* pattern

# Decorator Pattern

---

“In object-oriented programming, the decorator pattern is a design pattern that allows new/additional behaviour to be added to an existing object dynamically.”

- Wikipedia, “Decorator\_pattern”



# Two principal techniques

---

- Defined interface that all concrete objects and decorators adhere to
  - ▶ Often, decorators will implement an additional interface
- Duck-typing
  - ▶ Use method/property overloading in the decorator to *proxy* to the decorated object

```
interface Window
{
    public function isOpen ();
    public function open ();
    public function close ();
}
```

```
class StandardWindow implements Window {
    protected $_open = false;
    public function isOpen() {
        return $this->_open;
    }
    public function open() {
        if (!$this->_open) {
            $this->_open = true;
        }
    }
    public function close()
    {
        if ($this->_open) {
            $this->_open = false;
        }
    }
}
```

```
class LockedWindow implements Window
{
    protected $_window;

    public function __construct(Window $window) {
        $this->_window = $window;
        $this->_window->close();
    }

    public function isOpen() { return false; }

    public function open() {
        throw new Exception(
            'Cannot open locked windows');
    }

    public function close() {
        $this->_window->close();
    }
}
```

```

class LockedWindow
{
    protected $_window;
    public function __construct(Window $window)
    {
        $this->_window = $window;
        $this->_window->close();
    }
    public function isOpen() { return false; }
    public function __call($method, $args)
    {
        if (!method_exists($this->_window, $method))
        {
            throw new Exception('Invalid method');
        }
        return $this->_window->$method();
    }
}

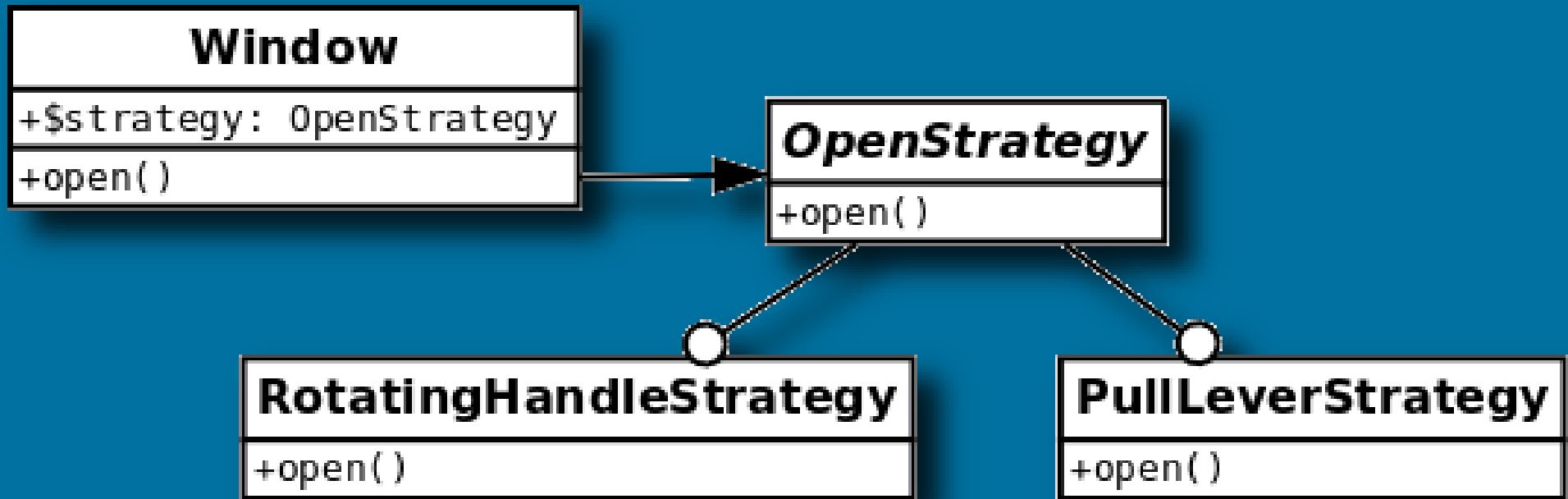
```

# Strategy Pattern

---

“A particular software design pattern, whereby algorithms can be selected at runtime.”

- Wikipedia, “Strategy\_pattern”



```
class Window
{
    public $strategy;

    public function open()
    {
        $this->$strategy->open();
    }
}
```

```
interface OpenStrategy
{
    public function open();
}
```

```
class RaiseStrategy implements OpenStrategy
{
    public function open() { }
}
```

```
class LeverStrategy implements OpenStrategy
{
    public function open() { }
}
```

# Creating your first decorator

# The Interface

---

- **Zend\_Form\_Decorator\_Interface**
  - ▶ `__construct($options = null);`
  - ▶ `setElement($element);`
  - ▶ `getElement();`
  - ▶ `setOptions(array $options);`
  - ▶ `setConfig(Zend_Config $config);`
  - ▶ `setOption($key, $value);`
  - ▶ `getOption($key);`
  - ▶ `getOptions();`
  - ▶ `removeOption($key);`
  - ▶ `clearOptions();`
  - ▶ `render($content);`

# The Interface

---

- What you really need to know about:
  - ▶ `render($content);`

# A simple decorator for text input

---

- Will render a label
- Will render a text input
- Will grab metadata from the element and use it to define the generated output

```

class My_Decorator_SimpleInput
    extends Zend_Form_Decorator_Abstract
{
    protected $_format = '<label for="%s">%s</label>'
                        . '<input id="%s" name="%s"
type="text" value="%s"/>';

    public function render($content)
    {
        $element = $this->getElement();
        $name     = htmlentities(
            $element->getFullyQualifiedName());
        $label   = htmlentities($element->getLabel());
        $id      = htmlentities($element->getId());
        $value   = htmlentities($element->getValue());

        $markup  = sprintf(
            $this->_format,
            $name, $label, $id, $name, $value);
        return $markup;
    }
}

```

```
$decorator = new My_Decorator_SimpleInput();  
$element = new Zend_Form_Element('foo', array(  
    'label' => 'Foo',  
    'belongsTo' => 'bar',  
    'value' => 'test',  
    'decorators' => array($decorator),  
));
```

```
$element = new Zend_Form_Element('foo', array(  
    'label' => 'Foo',  
    'belongsTo' => 'bar',  
    'value' => 'test',  
    'prefixPath' => array('decorator' => array(  
        'My_Decorator' => 'path/to/decorators/',  
    )),  
    'decorators' => array('SimpleInput'),  
));
```

```
<label for="bar[foo]">Foo</label>  
<input id="bar-foo" name="bar[foo]"  
      type="text" value="test"/>
```

# What's the point?

---

- Re-usable
- Automatic escaping
- Select the attributes you want to propagate and output at run-time

# Layering decorators

# The \$content argument

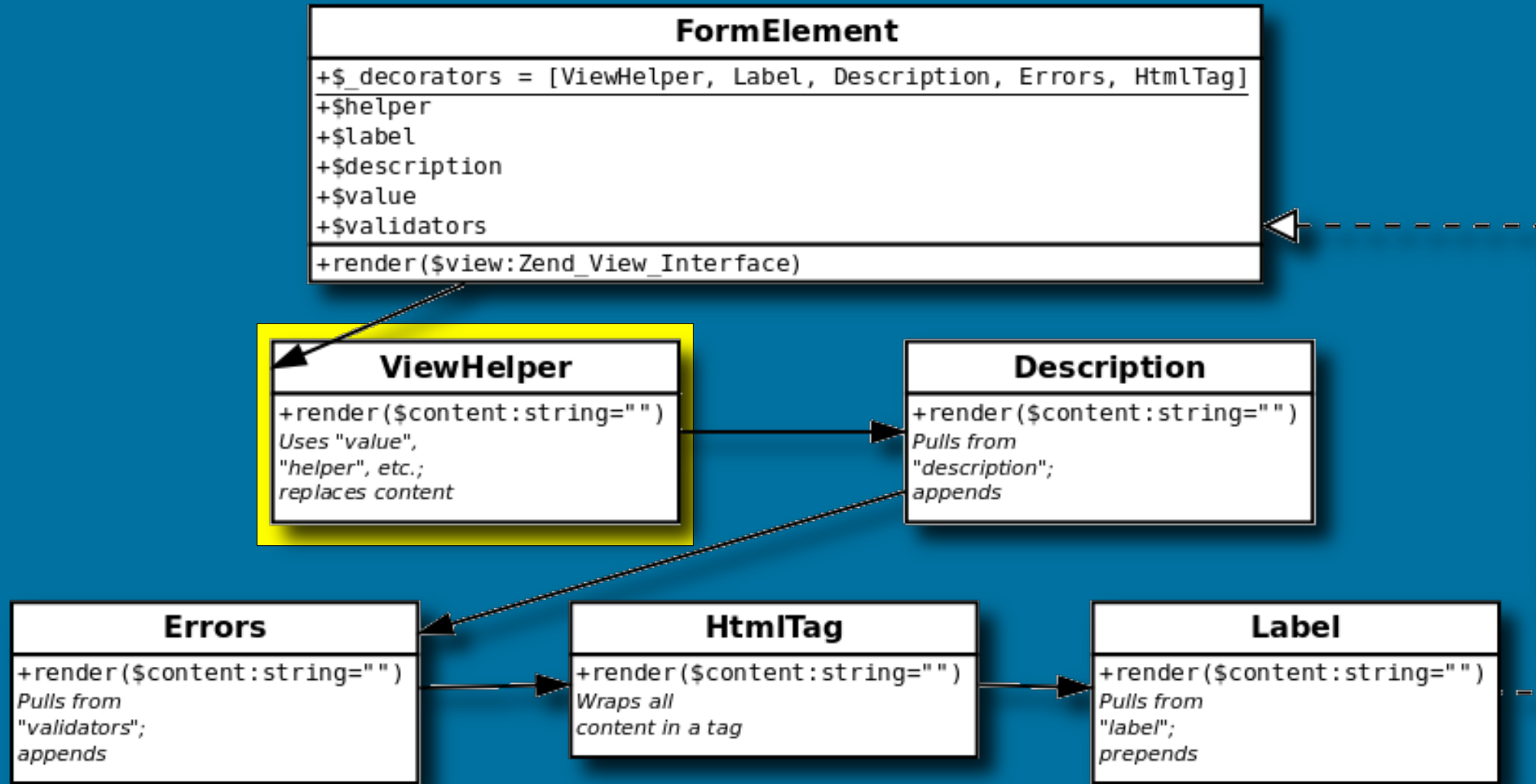
---

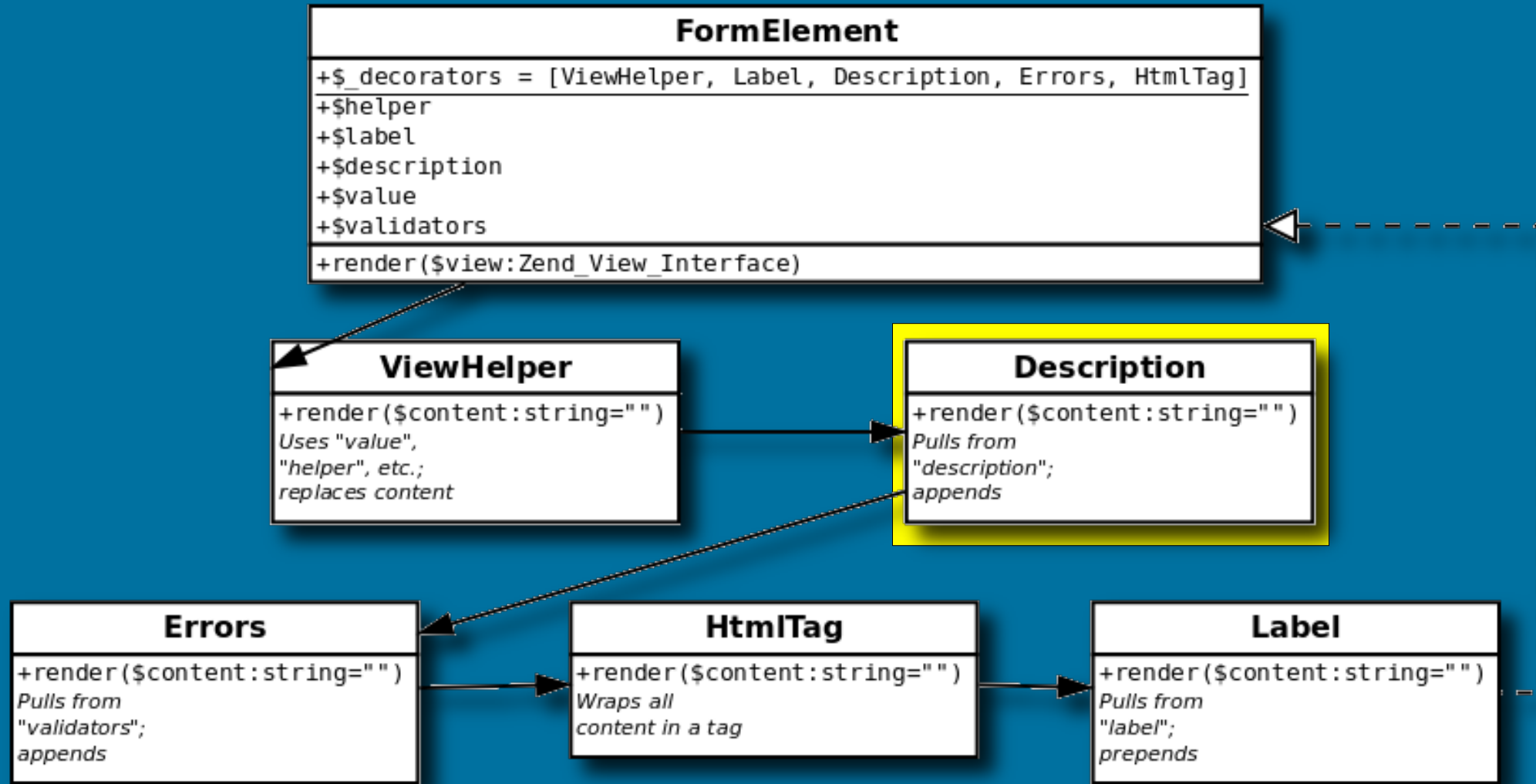
- Used to *aggregate* output from multiple decorators
- Individual decorators can specialize in creating output specific to pertinent element metadata
- Final output is the product of all decorators

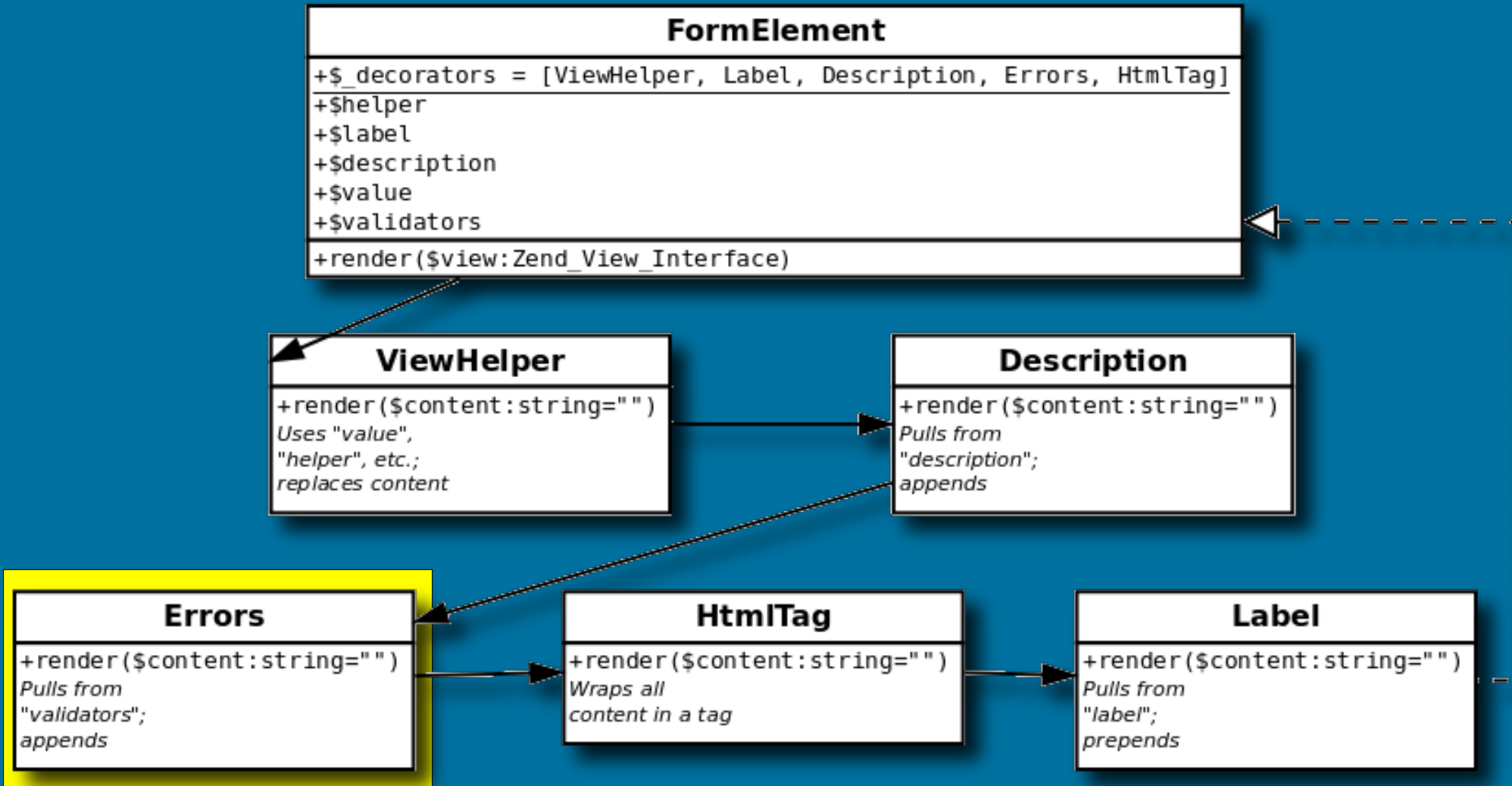
# Default decorators

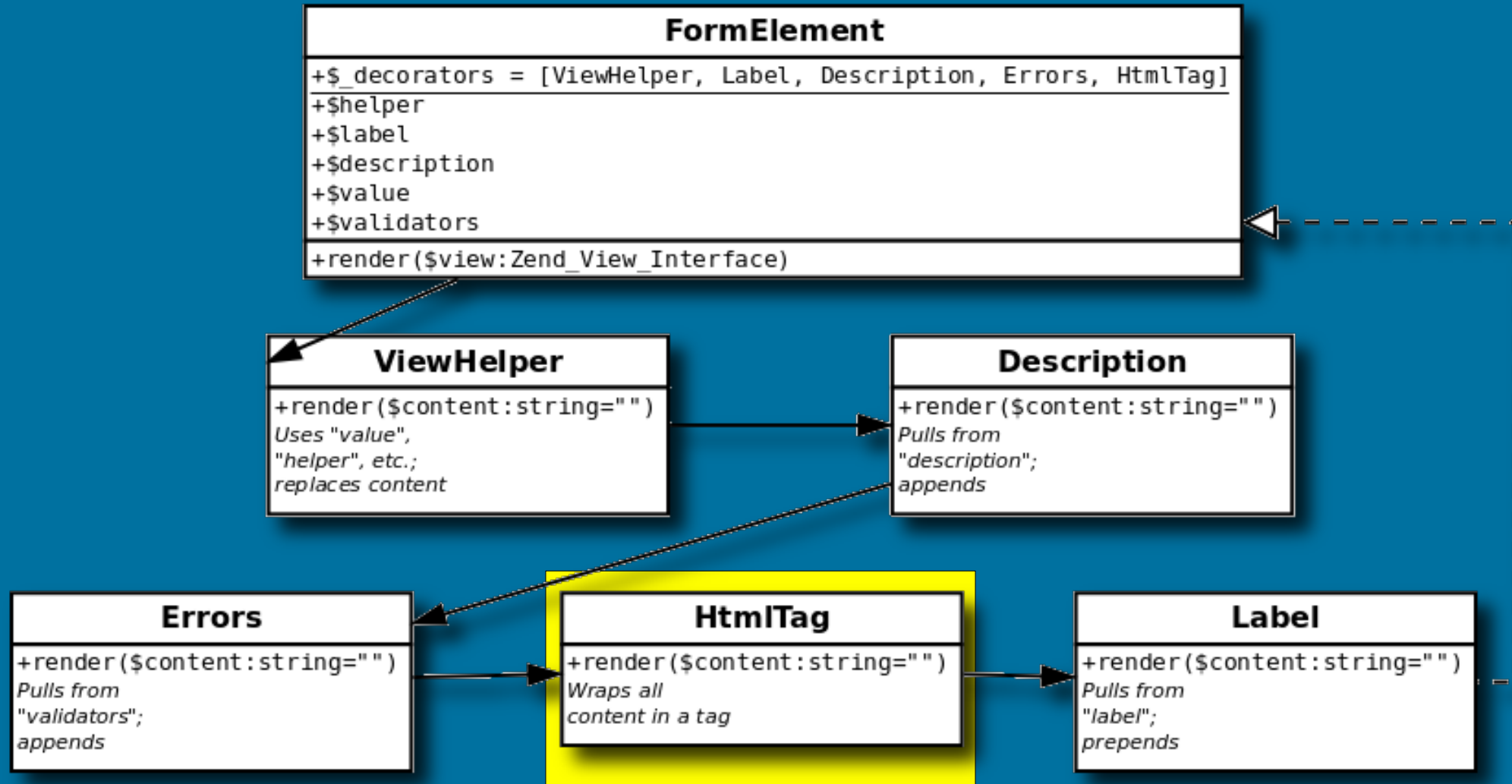
---

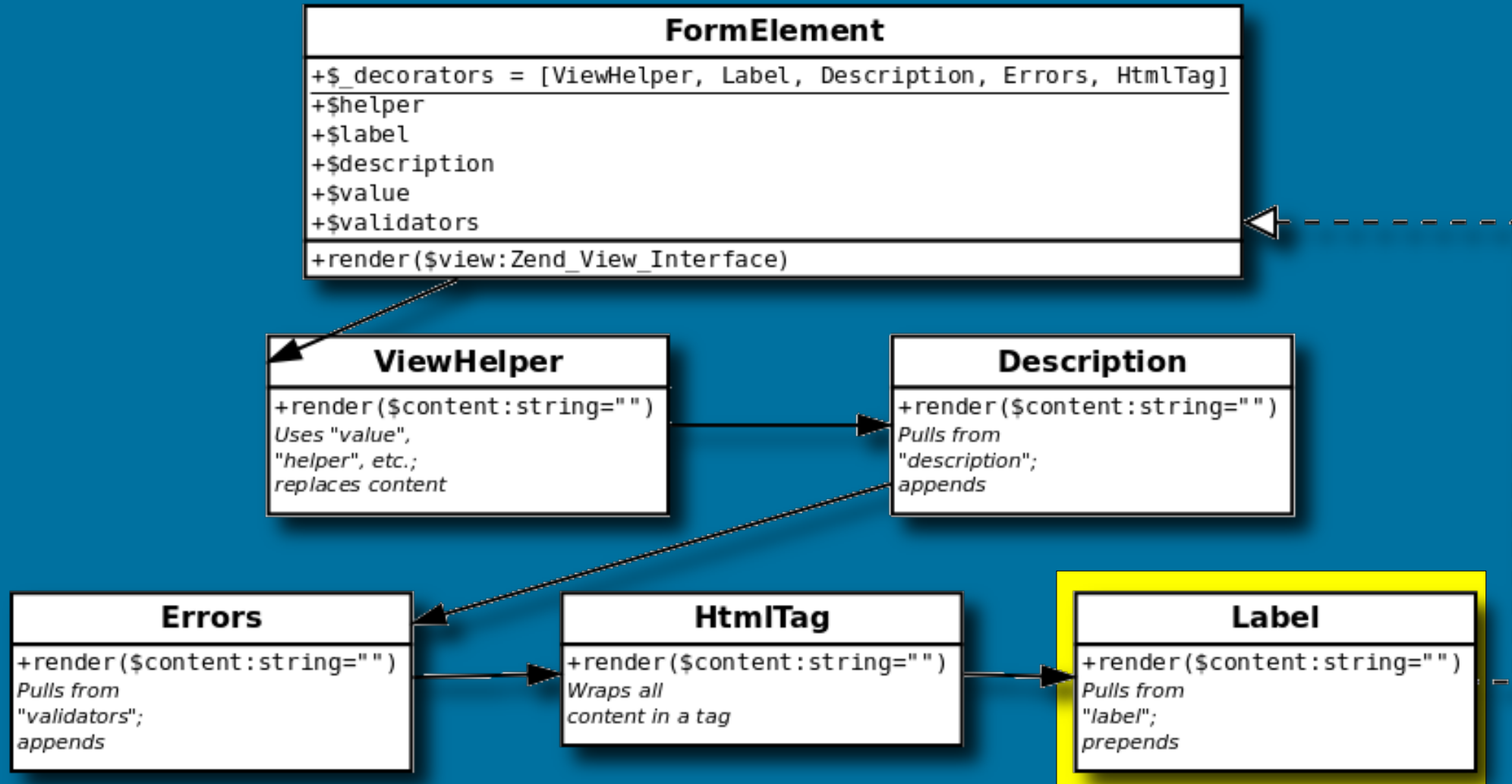
- **ViewHelper** - uses a view helper to render the form input itself
- **Description** - renders element desc, if any
- **Errors** - renders validation errors, if any
- **HtmlTag** - wrap all generated content in a tag
- **Label** - uses a view helper to render the element label; prepends to aggregate content











# How to layer

---

- Make use of the “placement” option
- Choose whether to:
  - ▶ Prepend the content provided
  - ▶ Append the content provided
  - ▶ Replace the content provided
- Have sane default placement; options allow passing in on override

```

class My_Decorator_SimpleInput
    extends Zend_Form_Decorator_Abstract
{
    protected $_format = '<input id="%s" name="%s"
type="text" value="%s"/>';

    public function render($content)
    {
        $element = $this->getElement();
        $name     = htmlentities(
            $element->getFullyQualifiedName());
        $id      = htmlentities($element->getId());
        $value   = htmlentities($element->getValue());

        $markup = sprintf($this->_format,
            $id, $name, $value);
        // determine placement here...
        return $markup;
    }
}

```

```
class My_Decorator_SimpleLabel
    extends Zend_Form_Decorator_Abstract
{
    protected $_format = '<label for="%s">%s</label>';

    public function render($content)
    {
        $element = $this->getElement();
        $id      = htmlentities($element->getId());
        $label   = htmlentities($element->getLabel());

        $markup = sprintf($this->_format, $id, $label);
        // handle placement here...
        return $markup;
    }
}
```

```
$placement = $this->getPlacement();  
$separator = $this->getSeparator();  
switch ($placement) {  
    case self::PREPEND:  
        return $markup . $separator . $content;  
    case self::APPEND:  
    default:  
        return $content . $separator . $markup;  
}
```

```
$element = new Zend_Form_Element('foo', array(  
    'label'           => 'Foo',  
    'belongsTo'     => 'bar',  
    'value'          => 'test',  
    'prefixPath'    => array('decorator' => array(  
        'My_Decorator' => 'path/to/decorators/',  
    )),  
    'decorators' => array(  
        'SimpleInput',  
        'SimpleLabel',  
    ),  
));
```

```
$element = new Zend_Form_Element('foo', array(  
    'label'           => 'Foo',  
    'belongsTo'     => 'bar',  
    'value'          => 'test',  
    'prefixPath'    => array('decorator' => array(  
        'My_Decorator' => 'path/to/decorators/',  
    )),  
    'decorators' => array(  
        'SimpleInput'  
        array('SimpleLabel', array(  
            'placement' => 'append'))),  
    ),  
));
```

# Rendering decorators individually

# Why?

---

- Quite simply, sometimes markup is too complex to assemble via layering
- Your designers may want to have more visibility into what is rendered

# How?

---

## Easy:

```
$decorator = $element->getDecorator(  
    'SimpleInput');  
echo $decorator->render('');
```

## Easier:

```
echo $element  
    ->getDecorator('SimpleInput')  
    ->render('');
```

## Easiest:

```
echo $element->renderSimpleInput();
```

```
<div class="element">
<?php
echo $form->title->renderLabel()
    . $form->title->renderViewHelper();
if ($form->title->hasErrors()) {
    echo '<p class="error">There were
        errors validating</p>';
}
?>
</div>
```

```
<div class="element">
    <?php echo $form->dateOfBirth->renderLabel() ?>
    <?php echo $this->formText('dateOfBirth[day]',
        '', array('size' => 2, 'maxlength' => 2)) ?>
    /
    <?php echo $this->formText('dateOfBirth[month]',
        '', array('size' => 2, 'maxlength' => 2)) ?>
    /
    <?php echo $this->formText('dateOfBirth[year]',
        '', array('size' => 4, 'maxlength' => 4)) ?>
</div>
```

# Creating composite elements

# Example: Date of Birth

---

- Store the date internally as a timestamp
- For usability, ask user to enter year, month, and day discretely
- ... but typical decorators represent an element with a single form input

# Create an element

---

- Accept value as an integer
- Accept value as an array with year/month/day segments
- Accept value as a string
- Accessors for each date segment (year, month, and day)

```

class My_Form_Element_Date extends Zend_Form_Element_Xhtml {
    public function setValue($value) {
        if (is_int($value)) {
            $this->_value = new DateTime(
                date('Y-m-d', $value));
        } elseif (is_string($value)) {
            $this->_value = new DateTime($value);
        } elseif (is_array($value)) {
            $this->_value = new DateTime();
            $this->_value->setDate(
                $value['year'], $value['month'], $value['day']
            );
        } else {
            throw new Exception('Invalid date value provided');
        }

        return $this;
    }
    public function getDay() { }
    public function getMonth() { }
    public function getYear() { }
}

```

# Create a decorator

---

- Duck-type check the element
- Ensure we have a view
- Grab metadata from the element
- Pass metadata to existing view helpers to create markup

```

class My_Form_Decorator_Date
    extends Zend_Form_Decorator_Abstract
{
    public function render($content) {
        $element = $this->getElement();
        if (!$element instanceof My_Form_Element_Date) {
            return $content;
        }

        $view = $element->getView();
        if (!$view instanceof Zend_View_Interface) {
            return $content;
        }

        $day      = $element->getDay();
        $month    = $element->getMonth();
        $year     = $element->getYear();
        $name     = $element->getFullyQualifiedName();
        // ...
    }
}

```

```

class My_Form_Decorator_Date
    extends Zend_Form_Decorator_Abstract
{
    public function render($content) {
        // ...
        $params = array('size' => 2, 'maxlength' => 2);
        $yearParams = array(
            'size' => 4, 'maxlength' => 4);

        $markup = $view->formText(
            $name . '[day]', $day, $params)
            . ' / ' . $view->formText(
                $name . '[month]', $month, $params)
            . ' / ' . $view->formText(
                $name . '[year]', $year, $yearParams);

        // placement and return ...
    }
}

```

# Update the element

---

- Ensure the element knows where to find its decorators
- Specify the default decorators

```
class My_Form_Element_Date
    extends Zend_Form_Element_Xhtml
{
    // ...
    public function __construct(
        $spec, $options = null
    ) {
        $this->addPrefixPath(
            'My_Form_Decorator',
            'My/Form/Decorator',
            'decorator'
        );
        parent::__construct($spec, $options);
    }
    // ...
}
```

```

class My_Form_Element_Date
    extends Zend_Form_Element_Xhtml
{
    // ...
    public function loadDefaultDecorators() {
        if ($this->loadDefaultDecoratorsIsDisabled()) {
            return;
        }

        if (empty(
            $decorators = $this->getDecorators()
        )) {
            $this->addDecorator('Date')
                // ...
                ;
        }
    }
    // ...
}

```

# Use it!

---

```
$d = new My_Form_Element_Date('dateOfBirth');  
$d->setLabel('Date of Birth: ');  
    ->setView(new Zend_View());
```

```
// These are equivalent:
```

```
$d->setValue('20 April 2009');  
$d->setValue(array(  
    'year' => '2009',  
    'month' => '04',  
    'day' => '20')));
```

```
<dt id="dateOfBirth-label"><label  
    for="dateOfBirth" class="optional">  
    Date of Birth:  
</label></dt>  
<dd id="dateOfBirth-element">  
    <input type="text" name="dateOfBirth[day]"  
        id="dateOfBirth-day" value="20"  
        size="2" maxlength="2"> /  
    <input type="text" name="dateOfBirth[month]"  
        id="dateOfBirth-month" value="4"  
        size="2" maxlength="2"> /  
    <input type="text" name="dateOfBirth[year]"  
        id="dateOfBirth-year" value="2009"  
        size="4" maxlength="4">  
</dd>
```

# Review

# What we've covered

---

- What decorators are, and the approach Zend\_Form uses
- How to create your own decorators
- How to layer decorators
- How to render individual decorators selectively
- How to create elements consisting of composite input



The PHP Company

# Thank you!

Zend Framework: <http://framework.zend.com/>

My twitter/IRC nick: weierophinney

My blog: <http://weierophinney.net/matthew/>