

The Seattle PHP Meetup Group—4 June 2009

The Decorator Design Pattern

Richard Cook

`rcook@rprodev.com`

The Decorator pattern

- Two competing (but related) definitions:
 - ◆ Method for extending functionality of an existing *class* dynamically
 - ◆ Method for extending functionality of an existing *object* dynamically

The Decorator pattern

- Two competing (but related) definitions:
 - ◆ Method for extending functionality of an existing *class* dynamically
 - ◆ Method for extending functionality of an existing *object* dynamically
- Alternative to subclassing:
 - ◆ Subclassing adds behaviour at *compile time*
 - ◆ Decorator adds behaviour at *runtime*

Class vs. object

- Some dynamic programming languages (e.g. Python, Ruby) both are effectively the same since classes are first-class objects

Class vs. object

- Some dynamic programming languages (e.g. Python, Ruby) both are effectively the same since classes are first-class objects
- This is *not* the case with PHP

Class vs. object

- Some dynamic programming languages (e.g. Python, Ruby) both are effectively the same since classes are first-class objects
- This is *not* the case with PHP
- If we *own* the class definition it is usually easier to extend the class directly

Class vs. object

- Some dynamic programming languages (e.g. Python, Ruby) both are effectively the same since classes are first-class objects
- This is *not* the case with PHP
- If we *own* the class definition it is usually easier to extend the class directly
- Decorator useful, however, if:
 - ◆ Third-party component we don't want to touch
 - ◆ Instance of object from PHP extension
 - ◆ Multiple classes we want to extend in the same fashion

So, what about PHP?

- The latter, *object*, definition more useful to us

So, what about PHP?

- The latter, *object*, definition more useful to us
- This is what we'll concentrate on

An example

```
final class WordCounter
{
    private $sentence = '';

    public function __construct() {}

    public function sentence() {return $this->sentence;}

    public function setSentence($sentence) {$this->sentence = $sentence;}

    public function getWordCount() {return count(explode(' ', $this->sentence));}
}
```

- Class `WordCounter` is for “probable” use
- This is what the majority of users will consume
- Class is potentially `final` so it *cannot* be extended

A decorator base class

```
abstract class WordCounterDecorator
{
    private $inner;

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

    public function sentence() {return $this->inner->sentence();}

    public function setSentence($sentence) {$this->inner->setSentence($sentence);}

    public function getWordCount() {return $this->inner->getWordCount();}
}
```

- This is a simple pass-through
- Each method calls through to the inner object
- Generally we'll fully encapsulate `$inner` by making it `private`
- Class might be `abstract` to force it to be extended

A concrete decorator

```
class PhpCounter extends WordCounterDecorator
{
    public function __construct($inner) {parent::__construct($inner);}

    public function getWordCount()
    {
        $matches = array();
        return preg_match_all('/php/i', parent::sentence(), &$matches);
    }
}
```

- Specialized version
- Not as general as `WordCounter`
- Can create family of `WordCounter`-like classes from decorator

Usage pattern

```
$text = <<<END
This is a demonstration of the Decorator object-oriented
programming design pattern in PHP. This example uses new-style
PHP 5.x classes.
END;

$word_counter = new WordCounter;
$word_counter->setSentence($text);

// Most common scenario.
echo 'Text contains ' . $word_counter->getWordCount() . ' words.<br />';

$php_counter = new PhpCounter($word_counter);

// Less common scenario.
echo 'PHP appears ' . $php_counter->getWordCount() . ' times.<br />';
```

- Many other ways to implement decorator

About me...

- Multilingual web developer...
 - ◆ PHP
 - ◆ Python
 - ◆ Ruby
 - ◆ C#

About me...

- Multilingual web developer...
 - ◆ PHP
 - ◆ Python
 - ◆ Ruby
 - ◆ C#
- Keen open-source guy

About me...

- Multilingual web developer...
 - ◆ PHP
 - ◆ Python
 - ◆ Ruby
 - ◆ C#
- Keen open-source guy
- Senior developer at Microsoft Corporation

About me...

- Multilingual web developer...
 - ◆ PHP
 - ◆ Python
 - ◆ Ruby
 - ◆ C#
- Keen open-source guy
- Senior developer at Microsoft Corporation
- Graduate student at University of Washington

About me...

- Multilingual web developer...
 - ◆ PHP
 - ◆ Python
 - ◆ Ruby
 - ◆ C#
- Keen open-source guy
- Senior developer at Microsoft Corporation
- Graduate student at University of Washington
- `rcook@rprodev.com`