Practical PHP 5.3

Nate Abele 7.27.2010 NYPHP New York

Me

- Former lead developer, CakePHP
- Co-founder & current lead developer, Lithium
- Developing on 5.3 for ~2 years
- Twitter: @nateabele

PHP Renaissance

- Long-awaited features finalized and committed
- Resurgence of energy and interest in evolving the language
- Trunk == iteration on latest stable release

I Unicode

Anyway...

Little Things

- Performance
- Syntax
- Phar
- SPL classes
- ext/fileinfo
- ext/sqlite3
- ext/intl
- mysqlnd

Big Things

- Late static binding
- Namespaces
- Lambdas / closures

Late-static binding (LSB)

```php
class Base {
    public static function foo() {
        $subclass = get_called_class();
        // ...do some logic with $subclass...
    }
}

class Child extends Base {
    public static function bar() {
        return static::foo() + 1;
    }
}

class Grandchild extends Child {
    public static function foo() {
        $result = parent::foo();
        // ...do some logic with $result...
        return $result;
    }
}
```

Late-static binding (LSB)

- `static::` ~= `$this->`
- `get_called_class()` - the subclass that invoked the method
- `parent::` - same idea you're used to
- Always use `static::` or `parent::`, `self::` etc. breaks the chain

Late-static binding (LSB)

- Attributes still a little “broken”

```
class A {
    public static $message;
}
class B extends A {}
class C extends A {}

B::$message = 'WTF?';
echo C::$message; // echoes 'WTF?'
```

Late-static binding (LSB)

- Attributes still a little “broken”

```
class A {
    public static $message;
}
class B extends A {}
class C extends A {
    public static $message = 'Overridden';
}

B::$message = 'WTF?';
echo C::$message; // echoes 'Overridden'
```

Late-static binding (LSB)

- Attributes still a little “broken”

```
class A {
    public static $message;
}
class B extends A {
    public static $message = 'Overridden';
}
class C extends A {
}

A::$message = 'WTF?';
echo C::$message; // echoes 'WTF?'
```

Why?

- Properly architect stateless portions of apps
- Utility classes
- Immutable state

“Dynamic” statics

- `$foo->$bar()` vs. `Foo::bar()`

- ```
  $foo = 'Foo';
  $bar = 'bar';
  $foo::$bar();
  ```

- `__callStatic()`
  ```

“Dynamic” statics

```
class Dispatcher {
    public function run($url) {
        $parameters = Router::match($url);
        // ...
    }
}
```

“Dynamic” statics

```
class Dispatcher {

    public $router = 'Router';

    public function run($url) {
        $router = $this->router;
        $parameters = $router::match($url);
        // ...
    }
}
```

Namespaces

- Finally!
- What's up with the \ ?
- Actually "packages"

Namespaces

```php
namespace foo\bar;

class MyClass {
    // ...
}

$class = new foo\bar\MyClass();

// -- or --

use foo\bar\MyClass;
$class = new MyClass();
```

Namespaces

```php
use foo\bar;
$class = new bar\MyClass();
```

```php
namespace me;
$list = new SplDoublyLinkedList();
```

```php
namespace me;
$list = new \SplDoublyLinkedList();
```

```php
namespace me;
$class = 'foo\bar\MyClass';
$object = new $class();
```

PSR-0

http://groups.google.com/group/php-standards/web/psr-0-final-proposal

PSR-0

- Maps top-level vendor package name to filesystem location
- Maps file names 1:1 with class name

PSR-0

- `lithium\core\Libraries =>`
  `/path/to/classes/lithium/core/Libraries.php`

- `lithium\Libraries =>`
  `/path/to/classes/lithium/Libraries.php`

- `Lithium\Core\Libraries =>`
  `/path/to/classes/Lithium/Core/Libraries.php`

- `Lithium_Core_Libraries =>`
  `/path/to/classes/Lithium/Core/Libraries.php`

Original (PEAR) Draft

- Requires sub-packages
- Namespaces must be lower-cased and underscored

Original (PEAR) Draft

- `Lithium\Core`: ?
- `lithium\core`: Obviously a namespace
- Sub-packages promote reuse outside vendor libraries

SplClassLoader

- Concrete implementation of PSR-0
- PHP version:
  http://gist.github.com/221634
- C version:
  http://github.com/metagoto/splclassloader

Intermission

Ternary Shortcut

```php
// Old:
public function foo($parameter = null) {
    $parameter = $parameter ? $parameter : $this->_calculateDefault();
    // ...
}

// New:
public function foo($parameter = null) {
    $parameter = $parameter ?: $this->_calculateDefault();
    // ...
}
```

Phar

- `include 'phar:///path/to/file.phar/file.php';`
- Redistributable apps
- Plugins
- Application templates

Phar

```php
$archive = new Phar("/path/new_file.phar");

$archive->buildFromDirectory(
    '/path/to/my/app',
    '/\.(php|htaccess|jpg|png|gif|css|js|ico|json|ini)$/'
);

$archive->compress(Phar::GZ);
```

Lambdas & Closures

- Lambda: a function assigned to a variable
- Closure: a lambda, but bound to variables in the current scope
- This is an extremely big deal

Lambdas & Closures

```php
$names = array(
    'Nate Abele', 'David Coallier', 'Cap\'n Crunch'
);
$split = array_map(
    function($name) {
        list($first, $last) = explode(' ', $name);
        return compact('first', 'last');
    },
    $names
);

// Result:
array(
  array('first' => 'Nate', 'last' => 'Abele'),
  array('first' => 'David', 'last' => 'Coallier'),
  array('first' => 'Cap\'n', 'last' => 'Crunch')
)
```

Lambdas & Closures

```php
$names = array('Nate Abele', 'David Coallier', /* ... */);
$filter = array('Nate', 'David');

$mapper = function($name) use ($filter) {
    list($first, $last) = explode(' ', $name);
    if (!in_array($first, $filter)) {
        return null;
    }
    return compact('first', 'last');
};
$filtered = array_map($mapper, $names);
```

Lambdas & Closures

```php
$findActive = function($object) use (&$findActive) {
    if ($object->isActive) {
        return $object;
    }
    if ($object->children) {
        foreach ($object->children as $child) {
            return $findActive($child);
        }
    }
};
```

Lambdas & Closures

```
class SomeWebService {

    public function call($data) {
        $request = new HttpRequest();
        // Configure $request...
        $result = $request->send();
        // Do some processing...
        // Update $request for next operation...
        $final = $request->send();
        // Post-process $final to extract some value
        return $someExtractedValue;
    }
}
```

Lambdas & Closures

```
class SomeWebService {

    public function call($data, $pre = null, $post = null) {
        $request = new HttpRequest();
        // ...
        if ($pre) {
            $request = $pre($request);
        }
        $result = $request->send();

        if ($post) {
            $result = $post($result);
        }
        // ...
        $final = $request->send();
        // ...
    }
}
```

Lambdas & Closures

```
class Database {

    public $callThisOnEveryRead;

    public function read($query) {
        if ($callback = $this->callThisOnEveryRead) {
            $callback($query);
        }
        // ...
    }
}
```

Lambdas & Closures

```
class Database {

    public $callThisOnEveryRead;

    public function read($query) {
        if ($callback = $this->callThisOnEveryRead) {
            $result = $callback($query)
            if ($result !== null) {
                return $result;
            }
        }
        // ...
    }
}
```

No `$this`

Referential Transparency

- Only needs parameters to calculate a return value (i.e. no `$_*`, `$this` or `date()`, etc.)
- Doesn't produce any side-effects
  - No modifications outside current scope
  - No references
  - No `echo`, `headers()`, etc.

lithium\util\collection\Filters

Method Filters

```
class Database {
    public function read($query) {
        // ... run the query ...
        return $result;
    }
}
```

Method Filters

```
class Database {
    public function read($query) {
        $method = function() {
            // ... run the query ...
            return $result;
        };
    }
}
```

Method Filters

```
class Database {
    public function read($query) {
        $method = function($query) {
            // ... run the query ...
            return $result;
        };
        return $method($query);
    }
}
```

Method Filters

```
class Database extends \lithium\core\Object {

    public function read($query) {
        $method = function($self, $params) {
            // ... run the query ...
            return $result;
        };
        return $this->_filter(
            __METHOD__, compact('query'), $method
        );
    }
}
```

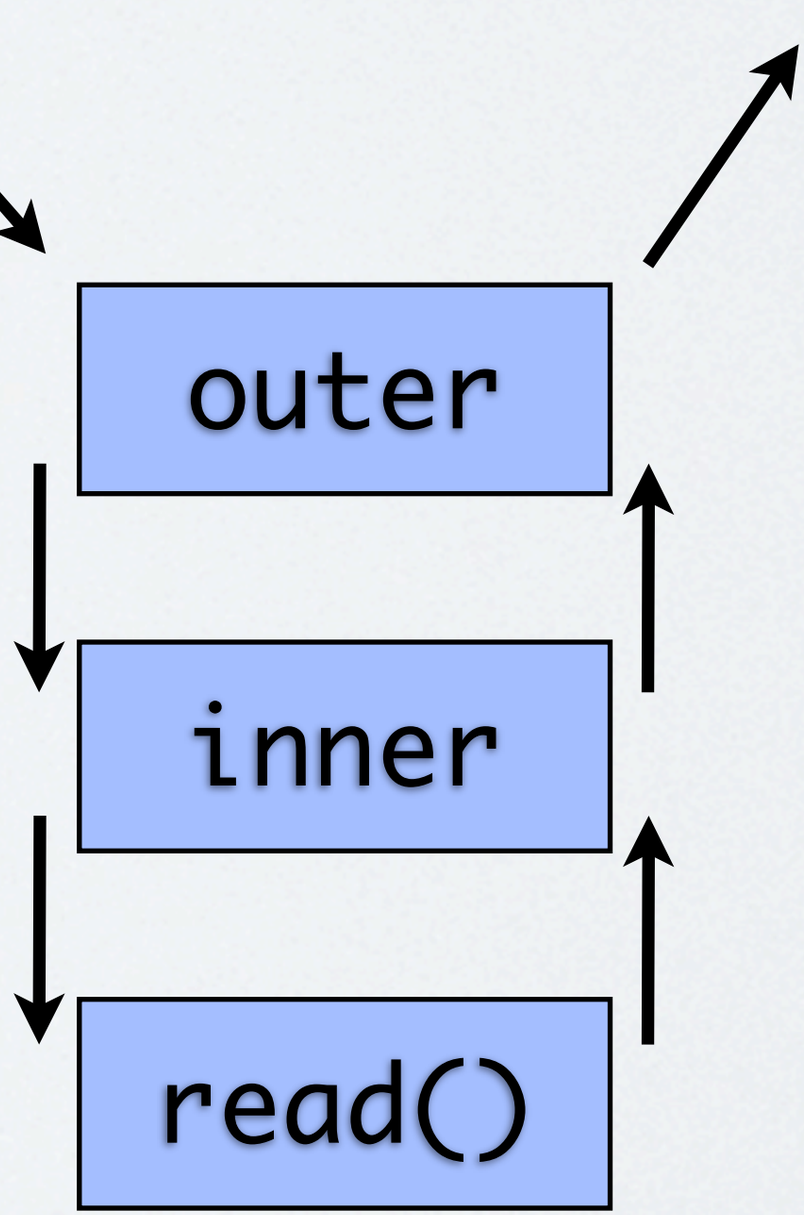
Method Filters

```php
$database = new Database($connectionDetails);

$database->applyFilter('read', function($self, $params, $chain) {
    $key = md5(serialize($params['query']));
    if ($result = Cache::read($key)) {
        return $result;
    }

    $result = $chain->next($self, $params, $chain);
    Cache::write($key, $result);
    return $result;
});
```

lithium\util\collection\Filters



AOP in a Nutshell

- Obliviousness
- Eliminates boilerplate code per class relationship
- Centralized configuration of concerns

Thanks!

- nate.abele@gmail.com
- @nateabele
- http://lithify.me/