

A Gentle Introduction to Object Oriented PHP

Central Florida PHP

A Gentle Introduction to Object Oriented PHP

- OOP: A Paradigm Shift
- Basic OO Concepts
- PHP4 vs. PHP5
- Case Study: Simplifying Requests
- Homework
- Suggested Reading

OOP: A Paradigm Shift

Procedural Code vs. Object Oriented Code

- Procedural code is *linear*.
- Object oriented code is *modular*.

Procedural Code vs. Object Oriented Code

```
mysql_connect();  
mysql_select_db();
```

```
$sql = "SELECT name FROM users ";  
$sql .= "WHERE id = 5 ";
```

```
$result = mysql_query($sql);  
$row = mysql_fetch_assoc($result);
```

```
echo $row['name'];
```

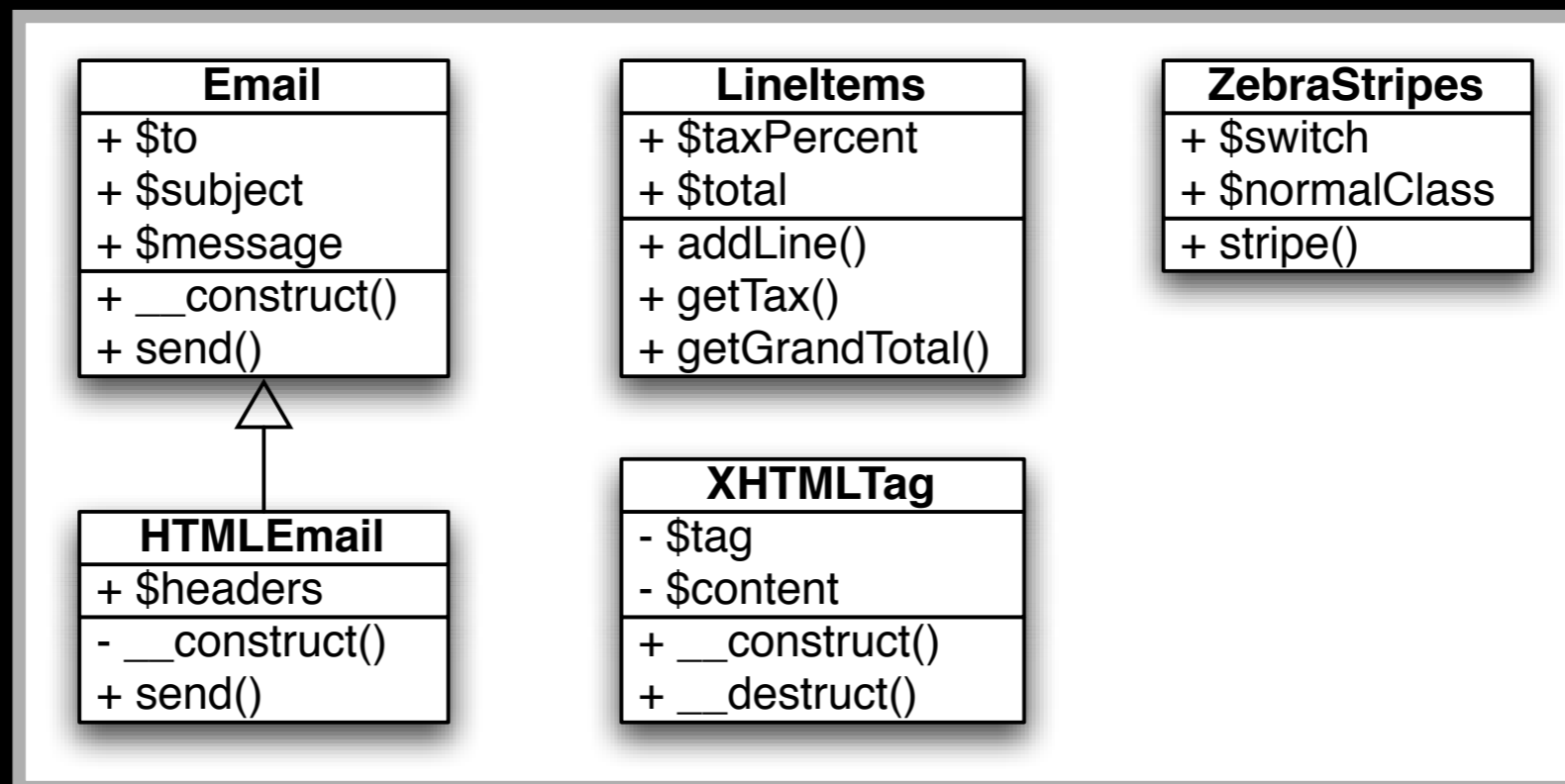
Procedural Code vs. Object Oriented Code

```
$User = new User;  
$row = $User->find('id=5', 'name');  
echo $row['name'];
```

Procedural Code vs. Object Oriented Code

```
1. <?php  
2.  
3. echo "Hello, World!";  
4.  
5. ?>
```

Procedural Code vs. Object Oriented Code



When Procedural is Right

- Small bite-sized chunks
- Lightweight “Front End” code
- Sequential scripts

When OO is Right

- Large, enterprise-level applications.
- “Back-end” related code for heavy lifting.

The OO Mindset

- Flexibility is essential
 - “Code for an interface, not an implementation.”
- Everything has a pattern
- Everything is an Object
- Iterate fast. Test often.
- Smaller is better.

Basic OO Concepts

Classes & Objects

- Classes are *templates*

```
class Email {  
    var $to;  
    var $from;  
    var $subject;  
    var $message;  
}
```

Classes & Objects

- Objects are *instances*

```
$one    = new Email;  
$two    = new Email;  
$three  = new Email;  
$four   = new Email;  
$five   = new Email;  
$six    = new Email;
```

Classes & Objects

- Classes are *templates*
 - A structured “shell” for a custom data type.
 - A class never executes.
- Objects are *instances*
 - A “living” copy of a class.
 - A class executes (if you tell it to).

Properties & Methods

- Properties describe an object

```
$one = new Email;  
$one->to = 'mike@example.com';  
$one->from = 'joe@example.com';  
$one->subject = 'Test';  
$one->message = 'Can you see me?';
```

Properties & Methods

- Methods are actions an object can make

```
$one->setFormat('text/plain');  
$one->addAttachment('virus.exe');  
$one->send();
```

Properties & Methods

- Properties describe an object
 - Variables that are *local* to an object
- Methods are actions an object can make
 - Functions that are *local* to an object
 - Have full access to any member in its scope (aka: `this`).

Constructors

- One of many “magic” methods that PHP understands.
- Runs immediately upon object instantiation.

```
class Gump {  
    function __construct() {  
        echo "Run Forrest! Run!";  
    }  
}
```

Constructors

- Parameters may also be passed to a constructor during instantiation

```
class Person {  
    function __construct($name) {  
        echo "Hi. I am $name.";  
    }  
}
```

```
$me = new Person('Mike');
```

Destructors

- Another “magic” method understood by PHP (there are lots of these guys, btw).
- Automatically called when an object is cleared from memory

```
class Kaboom {  
    function __destruct() {  
        echo “Cheers. It’s been fun!”;  
    }  
}
```

Inheritance

- Establishes a hierarchy between a parent class and a child class.
- Child classes *inherit* all visible members of its parent.
- Child classes *extend* a parent class' functionality.

Inheritance

```
class Parent {  
    function __construct() {  
        echo "I am Papa Bear.";  
    }  
}  
class Child extends Parent {  
    function __construct() {  
        echo "I am Baby Bear.";  
    }  
}
```

Inheritance

- When a child class defines a method that exists in a parent class, it *overrides* its parent.
- A parent member may be accessed via the “scope resolution operator”

```
parent::memberName()  
parent::$memberName;
```

Finality

- Inheritance may be prevented by declaring a class or method “final”
- Any attempt to extend or override a final entity will result in a fatal error
- Think before you use this

Visibility

```
class PPP {  
    public $a = 'foo';  
    private $b = 'bar';  
    protected $c = 'baz';  
  
    public function setB($newA) {  
        // code...  
    }  
}
```

Visibility

- Class members may be listed as
 - Public
 - Private
 - Protected

Visibility

- Public members are visible in from anywhere.
 - Global Scope
 - Any Class' Scope
- The `var` keyword is an alias for `public`

Visibility

- Private members are visible only to members of the same class.
- Viewing or editing from outside of `$this` will result in a parse error.

Visibility

- Protected members are visible within `$this` or any descendant
- Any public access of a protected member results in a parse error.

Static Members

- Members that are bound to a class, not an object.
- A static member will maintain value through *all instances* of its class.

Static Members

```
class Counter {  
    public static $i;  
    public function count() {  
        self::$i++;  
        return self::$i;  
    }  
}
```

```
echo Counter::count();  
echo Counter::count();
```

Static Members

- When referencing a static member, `$this` is not available.
 - From outside the class, `ClassName::$member`;
 - From inside the class, `self::$member`;
 - From a child class, `parent::$member`;

PHP4 vs. PHP5

PHP4 OOP

- No visibility
 - Everything is assumed public
- Objects passed by value, not by reference.

PHP5 OOP

- Completely rewritten object model.
- Visibility
- Objects passed by reference, not by value.
- Exceptions
- Interfaces and Abstract Classes
- ...

Case Study: Simplifying Requests

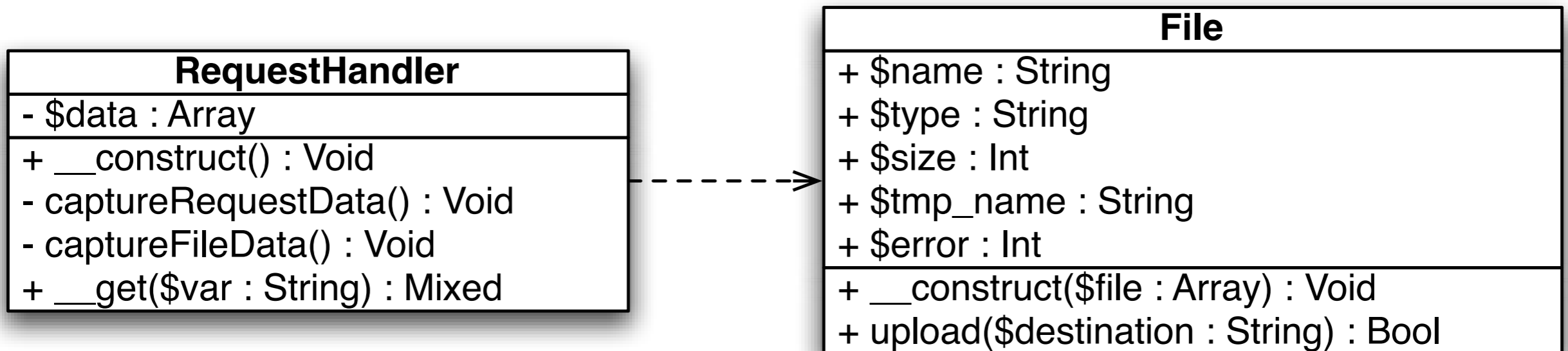
Understanding the Problem

- Request data is stored within several rather verbose superglobals
 - `$_GET['foo'], $_GET['bar']`
 - `$_POST['baz'], $_POST['bang']`
 - `$_FILES['goo']['tmp_name']`

Understanding the Problem

- Their naming convention is nice, but...
 - Associative arrays don't play well with quoted strings
 - Data is segregated over several different arrays (this is both good and bad)
 - Programmers are lazy

The UML



Homework

- Write an HTML generation library
 - Generates valid XHTML elements
 - Generates valid XHTML attributes
 - Knows how to self close elements w/no content
- Post your code to our Google Group
 - groups.google.com/group/cfphp

Suggested Reading

- **PHP 5 Objects, Patterns, and Practice**
Matt Zandstra, Apress
- **Object-Oriented PHP Concepts, Techniques, and Code**
Peter Lavin, No Starch Press
- **The Pragmatic Programmer**
Andrew Hunt and David Thomas, Addison Wesley
- **Advanced PHP Programming**
George Schlossngale, Sams