

WORKSHOP – LAMP SESSION 03

By Eddo Rotman



Now where were we?

- Had a little more syntax
- We started playing with PHP
- Had a sneak peak into PHP directives
- Discussed how to transfer data from the client to the server
- Discussed how to save persistent data between requests

So, what's the plan for today?

- Classes & Objects
- Zend Framework
 - As a framework
 - As a toolbox

Objects

```
<?php
class User {

    const NUMBER_OF_TEETH = 42;

    /**
     * @var string
     */
    private $name = 'dino';

    /**
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * @param string $name
     */
    public function setName($name) {
        $this->name = $name;
    }
}
```

Classes & Objects

- Accessing an object's methods and properties is done with the `->` operator

```
$dog->getName();  
$math->calculateSum(4,5,6);
```

- There are several reserved elements which allow accessing other parts of the class
 - `$this` – a pseudo variable which is a reference to the *object*
 - `self` – a reserved word which is a reference to the *class*
- Accessing an object's methods, static properties and constants with *self* is done with the scope resolution operator `::`

```
self::getItem();  
self::$foo; // $foo must be static  
self::NUM_OF_TEETH; // NUM_OF_TEETH is a class constant
```

Classes & Objects

- An object in PHP5 is *always* passed by reference, whether it is when it is passed to a function or when it is assigned anywhere

```
$newObj = $obj; // this is a reference to $obj  
$value = getValueFromObject($obj);  
setObjectsValue($obj);
```

- In order to “really” copy an object, we use the keyword *clone*

```
$anotherObj = clone $obj;
```

Constructors & Destructors

- Constructors and destructors are called automatically
- PHP uses a unified constructor `__construct()`
 - May accept as many parameters as needed
 - Will always return an object of the class
 - In PHP4 the constructor's name was the same as the class name. For backwards compatibility this will still work
- PHP uses a unified destructor `__destruct()`
 - May not accept any parameters
 - Will be called when:
 - Called explicitly
 - All references to the object are removed
 - When the process ends

Inheritance

- A class may inherit from only *one* parent class by using the *extends* keyword
- Methods of a parent class are *not* called from the child class automatically. If the parent's class method is needed, it can be called using the *parent* reserved word

```
<?php
class Animal {
    public function __construct($type) {
        $this->type = $type;
    }
}
class Dog extends Animal {
    public function __construct($name) {
        parent::__construct('Dog');
        $this->name = $name;
    }
}
```

Visibility

- The visibility of a method or a property can be defined by prefixing one of the following keywords to the method or properties
 - `public` – the resource can be accessed from anywhere
 - `protected` – the resource can be accessed from within the class where it is defined or its extending classes
 - `private` – the resource can be accessed from within the class where it is defined only
- Any method or property which doesn't have any of the above mentioned keywords attached to it is considered *public*

Visibility

- A child class may extend a property's visibility level

```
<?php
class Foo {
    protected $var;
}

class Bar extends Foo {
    public $var; // OK
}

class Baz extends Foo {
    private $var; // illegal
}
```

Scope Resolution

- The *static* keyword is used to declare a method which can be accessible without the need to instantiate the class
- A property declared as static can't be accessed with an instance of the class or the pseudo variable `$this`
- The scope resolution operator `::` can be used inside the class with `self` and `parent` and also to call constants, static properties and static methods outside the class
- When accessing items with the scope operator `::` from outside a class, use the class's name

Interface

- Interface classes specify a blueprint of which methods must be implemented in order for the class to satisfy some behavior
- All methods defined in an interface class must be public
- Defining an interface is done by using the *interface* keyword

```
interface Animal {  
    public function eat();  
    public function pee();  
    public function sleep();  
}
```

- Note that methods in an interface can not have a body

Interface

- Creating a class that supports the interface blueprint is done by using the *implements* keyword

```
class Dog implements Animal {  
    public function eat() { // eat food }  
    public function pee() { // pee on a tree }  
    public function sleep() { // sleep a lot }  
    public function getName() { // we can add more methods }  
}
```

- A class may implement more than one interface (unlike extending)

```
class Dog implements Animal, Mammal, ThaiFood {  
}
```

- An interface may extend another interface

```
interface Mammal extends Animal {  
}
```

Just a little more...

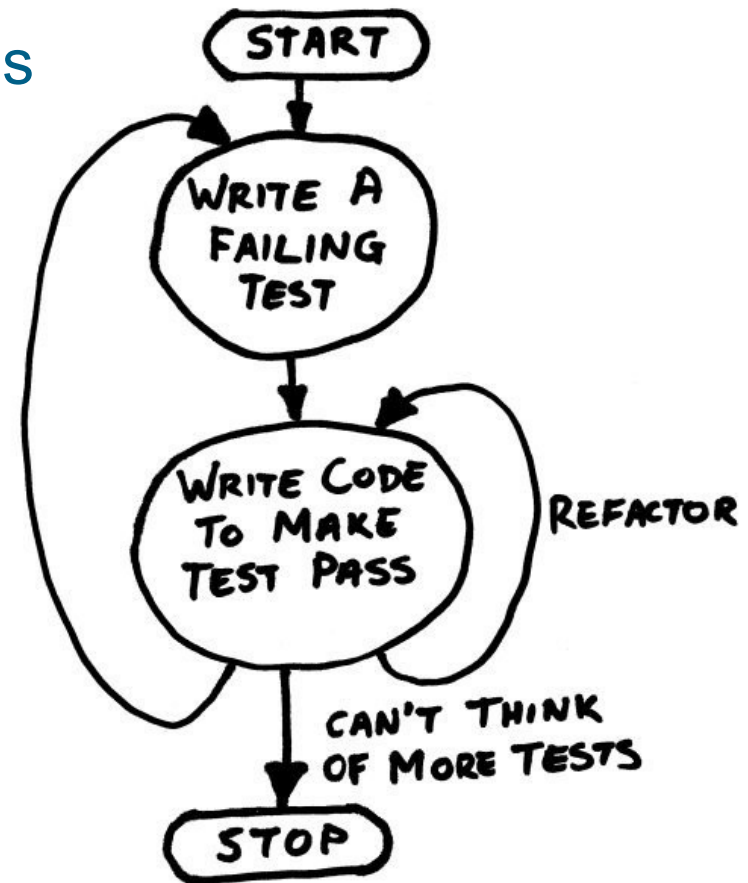
- Abstract
 - Abstract classes define the basic skeleton of a group of similar classes
 - Abstract classes may not be instantiated, but only extended
 - If a class has one abstract method or more, the class itself must be abstract
 - Any method marked as abstract in the parent class *must* be implemented in the child class
- Final
 - When it prefixes a class definition, it means the class may not be extended
 - When it prefixes to a method definition, it means the method may not be overridden

Operators

- Comparing objects
 - Two objects are considered equal `==` if
 - They have the same properties
 - Their properties have the same values
 - They are instances of the same class
 - Two object variables are considered identical `===` iff
 - They are pointing to the same object instance

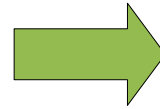
Test Driven Development

- Start by defining what your class is planned to do
- Write tests which match the definition
- Iteratively program in order to satisfy the tests
- Write more tests
- Write more code
- Repeat until done
- Refactor your code (and use your test cases as regression tests)



Sudoku Checker

2	6	4	5	8	1	3	9	7
1	3	5	7	9	4	2	8	6
7	8	9	6	3	2	4	5	1
4	2	8	9	6	3	7	1	5
3	5	7	1	4	8	6	2	9
9	1	6	2	5	7	8	4	3
5	9	3	4	2	6	1	7	8
6	4	1	8	7	9	5	3	2
8	7	2	3	1	5	9	6	4



```
2 6 4 5 8 1 3 9 7
1 3 5 7 9 4 2 8 6
7 8 9 6 3 2 4 5 1
4 2 8 9 6 3 7 1 5
3 5 7 1 4 8 6 2 9
9 1 6 2 5 7 8 4 3
5 9 3 4 2 6 1 7 8
6 4 1 8 7 9 5 3 2
8 7 2 3 1 5 9 6 4
```



264581397135794286789632451428963715357148629916257843593426178641879532872315964

PHP Built-in Classes

- PHP5 has many built-in classes which can simplify and secure our development
 - PDO
 - SimpleXMLElement
 - DOMDocument
 - Exception

Connecting to DB – old skool

```
<?php
$user = 'tautau';
$password = 'uTZyCfPK5uWb7eub';

$dbh = mysql_connect('localhost', $user, $password);
if (false !== $dbh) {
    if (false !== mysql_select_db('tautau')) {
        $result = mysql_query('SELECT * from Users');
        if (false !== $result) {
            while ($row = mysql_fetch_assoc($result)) {
                echo "{$row['firstName']} {$row['lastName']}<br />";
            }
        } else {
            echo "Error!: " . mysql_error() . "<br/>";
            die(2);
        }
    } else {
        echo "Error!: " . mysql_error() . "<br/>";
        die(3);
    }
    mysql_close();
} else {
    echo "Error!: " . mysql_error() . "<br/>";
    die(1);
}
```

Connecting to DB – new skool

```
<?php
$user = 'tautau';
$password = 'uTZyCfPK5uWb7eub';

try {
    $dbh = new PDO('mysql:host=localhost;dbname=tautau', $user, $password);
    foreach($dbh->query('SELECT * from Users') as $row) {
        echo "{$row['firstName']} {$row['lastName']}<br />";
    }
    $dbh = null; // destructor – close the connection
} catch (PDOException $e) {
    echo "Error!: " . $e->getMessage() . "<br/>";
    die(1);
}
```



Zend Framework

- Two major uses of Zend Framework
 - An MVC framework
 - An independent toolbox
- Zend Framework aims at adopting a *use-at-will* philosophy, which means less dependencies between its components
 - You don't have to use the entire framework
 - You don't have to build your application in a specific way



<http://framework.zend.com>

<http://devzone.zend.com/tag/Zend%20Framework>

Zend Framework as a Toolbox

- Zend Framework has many tools which may be used without using the MVC framework
- Most of them have very little dependencies, so you don't have to include the whole Zend Framework in your project
- Zend Framework is installed as part of Zend Server and is available by default in the system's include path

Zend_Db

```
<?php
require_once 'Zend/Db.php';

// Connecting to MySQL through MySQLi
$config = array(
    'adapter' => 'Pdo_Mysql',
    'hostname' => 'localhost',
    'dbname' => 'tautau',
    'username' => 'tautau',
    'password' => 'uTZyCfPK5uWb7eub',
);

try {
    $dbh = Zend_Db::factory($config['adapter'], $config);
    foreach($dbh->fetchAll('SELECT * from Users') as $row) {
        echo "{$row['firstName']} {$row['lastName']}<br />";
    }
    $dbh = null; // destructor - close the connection
} catch (Zend_Db_Exception $e) {
    echo "Error!: " . $e->getMessage() . "<br/>";
    die(1);
}
```

Zend_Http_Client

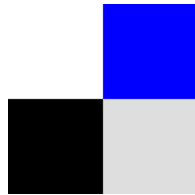
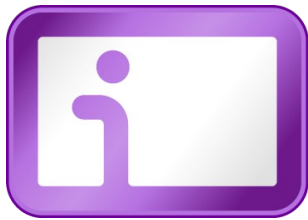
```
<?php
require_once 'Zend/Http/Client.php';
// Set up the client and enable cookie persistence
$client = new Zend_Http_Client('http://www.example.com');
$client->setCookieJar();

// Set the login parameters
$client->setParameterPost(array(
    'username' => 'eddo',
    'password' => 'myPassword'
));

// Send a POST request
$response = $client->request(Zend_Http_Client::POST);

if ($response->isSuccessful()) {
    // Send a GET request to fetch the restricted content
    $client->resetParameters();
    $client->setUri('http://www.example.com/restricted');
    $response = $client->request(Zend_Http_Client::GET);
    // do something with the response
}
```

Zend_Service



Zend Framework as a Framework

Model View Controller (MVC)

A software architecture that separates the components of the application: the model represents the business logic or data; the view represents the user interface; and the controller manages user input or, in some cases, the application flow.

(www.ibm.com)

- Model – the “stuff” you are using in your application (e.g. data handling, web services, feeds, algorithms)
- View – the way your application is displayed
- Controller – manages the request environment and flow control of the application

Zend Framework as a Framework

- Front Controller
 - Single entry point to handle all the requests
 - Delegates the requests to Action Controllers
 - Returns responses
- Action Controller
 - A unit of the application which handles a group of action, usually grouped together either by topic (UserManageController) or by viewable section (AdminSectionController)
- Action
 - A method in an Action Controller which handles a specific action (can be an action explicitly or a display action)

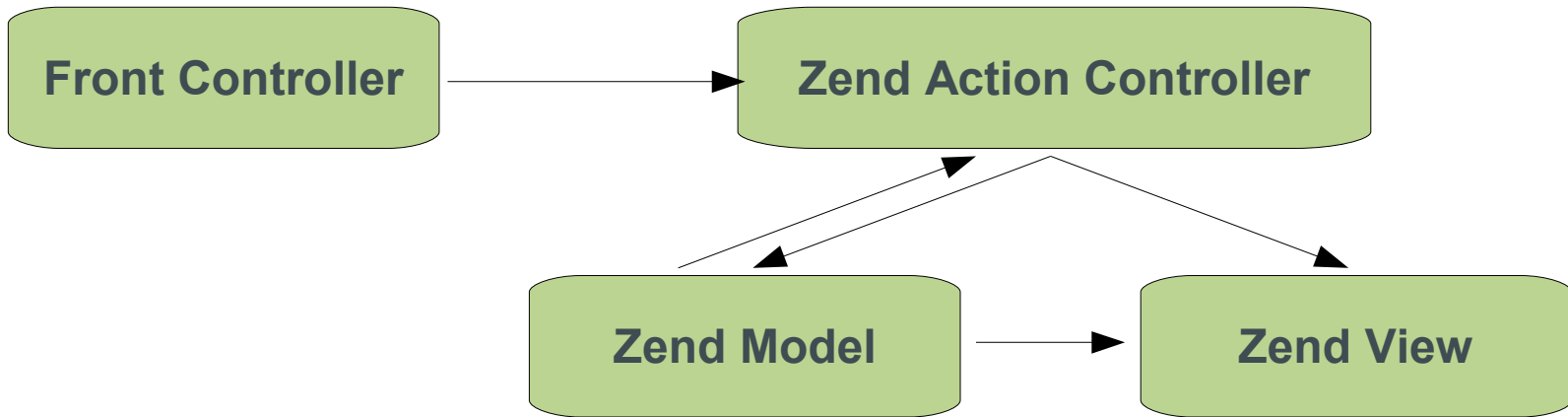
Zend Framework as a Framework

- Note that web MVC is a little different from application MVC as the View can not interact back with the Model unless it goes through the controller
- Naming standards – read the manual...
- There are many configuration options for Zend Framework projects and many project suggested structures



If you follow the ZF quick start, keep in mind it assumes your project is in the root of your `document_root`

Zend Framework as a Framework



- The Controllers pass and requests data from Models
- The Controllers pass data to Views
- The Views requests data from Models

Creating Your 1st ZF Application

- If you choose not to put your ZF project in your document root
 - Create a new PHP Project in your Zend Studio
 - Extract the contents of the Quick Start archive file into your PHP project folder
 - Edit your default site's configuration file (sites-available/default)
 - Make sure that under the document root's directory block AllowOverride is All
 - Create an Alias between some project name and the path to your public folder
 - Edit your public/.htaccess
 - Add a RewriteBase line using the same project name as before

Creating Your 1st ZF Application

- Test that accessing your project with and without specifying controller / action works
- Remove the unnecessary lines from the .htaccess (all until the comment starting with “The rules below”)
- Replace all short tags in the application with full tags
 - <?= → <?php echo (11 instances)
 - <? → <?php (5 instances)
- Start playing



<http://framework.zend.com/wiki/display/ZFDEV/Configuring+Your+URL+Rewriter>

Creating Your 1st ZF Application

```
DocumentRoot /var/www/  
Alias /tauproj /var/www/taufw/public  
<Directory />  
    Options FollowSymLinks  
    AllowOverride None  
</Directory>  
<Directory /var/www/>  
    Options Indexes FollowSymLinks MultiViews  
    AllowOverride All  
    Order allow,deny  
    allow from all  
</Directory>
```

```
# .htaccess  
RewriteEngine On  
RewriteBase /tauproject  
RewriteCond %{REQUEST_FILENAME} -s [OR]  
RewriteCond %{REQUEST_FILENAME} -l [OR]  
RewriteCond %{REQUEST_FILENAME} -d  
RewriteRule ^.*$ - [NC,L]  
RewriteRule ^.*$ index.php [NC,L]
```

Note: the prefixing / was removed

Questions?