Object Oriented Programming in PHP 5

Why OOP?

- Enhanced Flexibility
  - Abstraction
  - Modularization
  - Encapsulation
  - Extensibility
- Reuse of Code

SALES EMOTION

Cost of OOP

- Speed
  - OOP is slow compared to procedural code
    - Overhead in Initialization and execution
- Memory usage
- Verbosity
  - More Code even for small applications

When to use OOP

- OOP is a tool – not a religion ;)
  - Can be combined with procedural code
  - Mind the costs
- XML DOM
- SOAP
- MySQLi

SALES EMOTION

History

- PHP 3.x
  - Basic OOP Structure available
- PHP 4.x
  - Enhanced but still limited OOP
- PHP 5.x
  - Reimplemented, Java-like OOP Support

SALES EMOTION

OOP with PHP 5.x

```
class  MyClass {
      public $data;
      private $private;
      static $static=1;
      const constant = 'value';

      public function  __construct() {
      }

      public function __destruct() {
      }

      protected function Action() {
      }

      private function hasMore() {
      }

      public function show() {
      }
}
```

- keyword **class**
  - __construct() for constructor
    - old method still supported
  - __destruct() for destructor
    - newly available
- keyword **private**
  - only available within the class
- keyword **protected**
  - available within the class and all children

SALES EMOTION

OOP with PHP 5.x

```
class  MyClass {
      public $data;
      private $private;
      static $static=1;
      const constant = 'value';

      public function  __construct() {
      }

      public function __destruct() {
      }

      protected function Action() {
      }

      private function hasMore() {
      }

      public function show() {
      }
}
```

- keyword **public**
  - generally available
    - identical to **var** from PHP 4.x
- keyword **static**
  - Access MyClass::$static
    - can be combined with public, private or protected
- keyword **const**
  - Access via MyClass::constant
    - comparable to define()
    - can be combined with public, private or protected

Extending classes

```
class  MyClass {
      public $data;

      public function __construct() {
            $this->data='Hello world!';
      }
}


class Second extends MyClass {
      private $some;

      public function __construct() {
            $this->some='Second!';
            parent::__construct();
      }
}
```

- keyword **extends**
  - identical to PHP 4.x
  - if required, parent constructors must be called ***manually***

SALES EMOTION

Final classes & functions

```
final class  MyClass {
      public $data;

      public function __construct() {
            $this->data='Hello world!';
      }
}


class SomeClass {

      final public function foo() {
      }

}
```

- keyword **final**
  - available on class
    - prohibits use of “extends”
  - available on function
    - prohibits overriding in children

SALES EMOTION

Abstract classes & functions

```
abstract class  MyClass {
      public $data;

      public function __construct() {
            $this->data='Hello world!';
      }

      abstract public function foo(){
      }
}


class SomeClass extends MyClass {

      public function foo() {
      }

}
```

- keyword **abstract**
  - use on class
    - prohibits instantiation, enforce extends
  - use on function
    - enforce overriding in extended classes

SALES EMOTION

Interface & implements

```
interface  Template {

      public function foo($var);
}
```

```
class MyClass implements Template {

      public function foo($var){
            echo $var;
      }
}
```

```
class MyClass extends foobar
implements Template {

      public function foo($var){
            echo $var;
      }
}
```

- keyword **interface**
  - predefine a class structure
    - cannot be instantiated
    - functions must be implemented in the given way (e.g. parameters)
- keyword **implements**
  - create class based on interface
    - must implement all interface functions
    - can be combined with extends

Magic methods

```
class MyClass  {
      private $data;

      public function __get($var) {
            return $this->data[$var];
      }

      public function __set($var,$val) {
            $this->data[$var]=$val;
      }

      public function __call($fn,$arg) {
            echo $fn.” called”;
      }

}



function __autoload($class) {
      require_once($class.'.php');
}
```

- method **__get()**
  - called on read access to vars
    - enables indirect access to private
- method **__set()**
  - called on write access to vars
- method **__call()**
  - overload method calls

SALES EMOTION

Magic methods

```
class MyClass  {
      private $data;

      public function __get($var) {
            return $this->data[$var];
      }

      public function __set($var,$val) {
            $this->data[$var]=$val;
      }

      public function __call($fn,$arg) {
            echo $fn.” called”;
      }

}



function __autoload($class) {
      require_once($class.'.php');
}
```

- method **__sleep()**
  - called on serialize()
    - prepare serialization
      - close database or open files
- method **__wakeup()**
  - called on unserialize()
    - re-open connections
- function **__autoload()**
  - called on undefined classes
    - require_once() class code

SALES EMOTION

From 4.x to 5.x

- PHP 5.x passes objects by reference
  - In 4.x this has to be enforced by &$inst
  - Use $foo = clone $inst; for a copy

Singleton in PHP 5.x

```
class MyClass  {
      private static $instance=null;

      public static function getInstance(){
            if (self::$instance==null) {
                  self::$instance= new MyClass;
            }
            return self::$instance;
      }
}


$obj = MyClass::getInstance();
```

- private static $instance
  - reference to object instance
- private static function
  - create new instance if required
  - return static $instance
- static call to getInstance()

SALES EMOTION