ICT 5 Web Development
Chapter 5. OOP in PHP

## Object-Oriented Programming (OOP)

- Object: Instance (occurrance) of a class
- Classes/Objects encapsulates their data (called attributes) and behaviour (called methods)
- Inheritance: Define a new class by saying that it's like an existing class, but with certain new or changed attributes and methods.
  - The old class: superclass/parent/base class
  - The new class: subclass/child/derived class

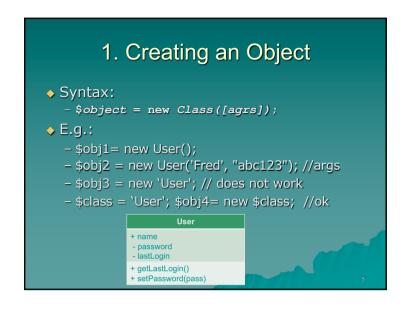
- 1

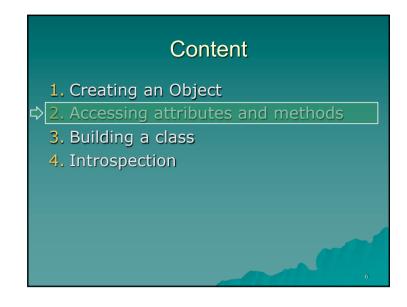
## PHP 5

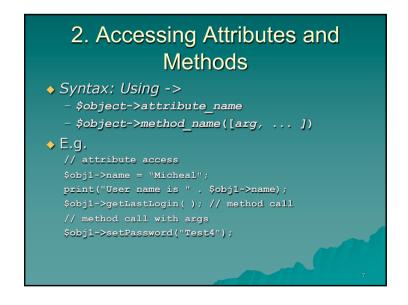
- Single-inheritance
- Access-restricted
- ◆ Overloadable
- ◆Object ~ pass-by-reference

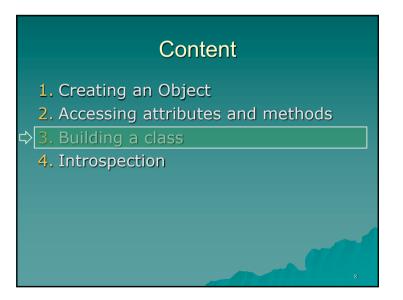
## Content

- $\Rightarrow$  1. Creating an Object
  - 2. Accessing attributes and methods
  - 3. Building a class
  - 4. Introspection

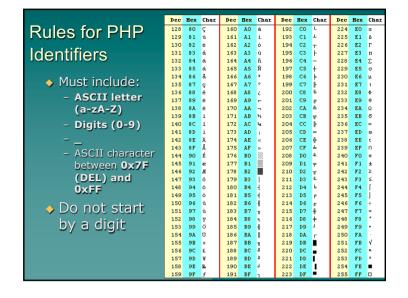


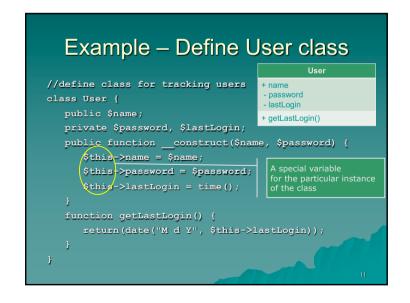






## 





## 

```
<?php
  class BaseClass {
     function __construct() {
        print "In BaseClass constructor\n";
     }
}

class SubClass extends BaseClass {
    function __construct() {
        parent::__construct();
        print "In SubClass constructor\n";
     }
}

$obj = new BaseClass();
$obj = new SubClass();
</pre>
```

```
class Counter {
    private static $count = 0;
    const VERSION = 2.0;
    function __construct() { self::$count++; }
    function __destruct() { self::$count---; }
    static function getCount() {
        return self::$count;
    }
    }
}
$c1 = new Counter();
print($c1->getCount() . "<br>yrint($c1->getCount() . "<br>n");
$c2 = new Counter();
print(Counter::getCount() . "<br>yrint($c1->getCount() . "<br>n");
$c2 = NULL;
print($c1->getCount() . "<br>yrint($c1->getCount() . "<br>yrint($c1->getCount() . "<br>yrint($c1->getCount() . "<br>yrint($c1->getCount() . "<br/>yrint($c1->getCount() . "<br/
```

### 3.3. Static & constant class members

- Static member
  - Not relate/belong to an any particular object of the class, but to the class itself.
  - Cannot use \$this to access static members but can use with self namespace or ClassName.
  - E.g.
    - count is a static attribute of Counter class
  - ♦ self::\$count Of Counter::\$count
- Constant member
  - value cannot be changed
  - can be accessed directly through the class or within object methods using the self namespace.

14

## 3.4. Cloning Object

- \$ \$a = new SomeClass();
- ♦ \$b = \$a;
- ♦ \$a and \$b point to the same underlying instance of SomeClass
- → Changing \$a attributes' value also make
   \$b attributes changing
- → Create a replica of an object so that changes to the replica are not reflected in the original object? → CLONING

## 3.4. Object Cloning

- ◆ Special method in every class: clone()
  - Every object has a default implementation for clone()
  - Accepts no arguments

\$b = clone \$a;

◆ Call cloning:

```
- $copy_of_object = clone $object;
-E.g.
$a = new SomeClass();
```

```
class ObjectTracker {
                                       Example -
   private static $nextSerial = 0:
   private $id, $name;
                                         Cloning
      $this->name = $name;
      this->id = ++self::$nextSerial:
                                               Hello world!
                                               1 Zeev's Object
      $this->id = ++self::$nextSerial;
                                               2 Another object
   function getName() { return($this->name); }
   function setName($name) { $this->name = $name; }
$ot = new ObjectTracker("Zeev's Object");
$ot2 = clone $ot; $ot2->setName("Another object");
print($ot->getId() . " " . $ot->getName() . "<br>");
print($ot2->getId() . " " . $ot2->getName() . "<br/>');
```

## 3.5. User-level overloading

- Overloading in PHP provides means dynamic "create" attributes and methods.
- The overloading methods are invoked when interacting with attributes or methods that have not been declared or are not visible in the current scope
  - inaccessible properties
- All overloading methods must be defined as public.

## 3.5.1. Attribute overloading

- void \_\_set (string \$name , mixed \$value)
  - is run when writing data to inaccessible attributes
- mixed \_\_get (string \$name)
  - $\,$  is utilized for reading data from inaccessible attributes
- ♦ bool isset (string \$name)
  - is triggered by calling isset() or empty() on inaccessible attributes
- void \_\_unset (string \$name)
  - is invoked when unset() is used on inaccessible attributes

Note: The return value of \_\_set() is ignored because of the way PHP processes the assignment operator. Similarly, \_\_get() is never called when chaining assignments together like this:

\$\frac{4}{3} = \frac{4}{3} \frac{1}{3} = \frac{8}{3} \frac{1}{3} = \fra

```
class PropertyTest {
   private $data = array();
                                     Example - Attribute overloading
   public $declared = 1;
                                                                Setting 'a' to '1'
   private $hidden = 2;
                                                               Getting 'a'
   public function __set($name, $value) {
   echo "Setting '$name' to '$value' <br/>';
                                                               Is 'a' set?
     this->data[$name] = $value;
                                                               bool(true) Unsetting 'a'
                                                               Is 'a' set?
                                                               bool(false)
        echo "Getting '$name' <br>";
        if (array key exists ($name, $this->data)) {
                                                               Getting 'hidden'
             return $this->data[$name];
                                                  $obj = new PropertyTest;
        echo "Is '$name' set?<br>";
        return isset($this->data[$name]); echo $obj->a."<br/>';
                                                  var_dump(isset($obj->a));
    public function __unset($name) {
    echo "Unsetting '$name' <br>';
                                                  unset($obj->a);
                                                  var_dump(isset($obj->a));
        unset($this->data[$name]);
                                                  echo "<br>";
   public function getHidden() {
                                                  echo $obj->declared."<br>"
        return $this->hidden;
                                                  echo $obj->getHidden()."<br>
                                                 echo $obj->hidden."<br>";
```

# 3.5.2. Method overloading mixed \_\_call (string \$name, array \$arguments) is triggered when invoking inaccessible methods in an object context mixed \_\_callStatic (string \$name, array \$arguments) is triggered when invoking inaccessible methods in a static context.

```
<?php
  class Foo {
    static $vals;
    public static function __callStatic($func, $args)
    {
        if (!empty($args)) {
            self::$vals[$func] = $args[0];
        } else {
            return self::$vals[$func];
        }
     }
    }
    Which would allow you to say:

</pre>

    Which would allow you to say:

    Foo::username('john');
        print Foo::username(); // prints 'john'
?>
```

## 3.6. Autoloading class

- Using a class you haven't defined, PHP generates a fatal error
- → Can use include statement
- → Can use a global function \_\_autoload()
  - single parameter: the name of the class
  - automatically called when you attempt to use a class PHP does not recognize

25

## Example - Autoloading class

```
//define autoload function
function __autoload($class) {
   include("class_".ucfirst($class).".php");
}
//use a class that must be autoloaded
$u = new User;
$u->name = "Leon";
$u->printName();
```

## 3.7. Namespace

- → ~folder, ~package
- Organize variables, functions and classes
- Avoid confliction in naming variables, functions and classes
- ◆ The namespace statement gives a name to a block of code
- From outside the block, scripts must refer to the parts inside with the name of the namespace using the :: operator

## 3.7. Namespace (2)

- You cannot create a hierarchy of namespaces
- → namespace's name includes colons as long as they are not the first character, the last character or next to another colon
- → use colons to divide the names of your namespaces into logical partitions like parent-child relationships to anyone who reads your code
- ◆ E.g. namespace hedspi:isl { ....

```
abstract class Shape {
   abstract function getArea();
}
abstract class Polygon extends Shape {
   abstract function getNumberOfSides();
}
class Triangle extends Polygon {
   public $base;
   public $height;
   public function getArea() {
      return(($this->base * $this->height)/2);
   }
   public function getNumberOfSides() {
      return(3);
   }
}
```

# 3.8. Abstract methods and abstract classes

- ◆ Single inheritance
- ◆ Abstract methods, abstract classes, interface (implements) like Java
- ◆You cannot instantiate an abstract class, but you can extend it or use it in an instanceof expression

```
class Rectangle extends Polygon {
   public $width; public $height;
   public function getArea() {
      return($this->width * $this->height);
   }
   public function getNumberOfSides() {
      return(4);
   }
}
class Circle extends Shape {
   public $radius;
   public function getArea() {
      return(pi() * $this->radius * $this->radius);
   }
}
class Color {
   public $name;
}
```

```
$myCollection = array();
$r = new Rectangle; $r->width = 5; $r->height = 7;
$myCollection[] = $r; unset($r);
$t = new Triangle; $t->base = 4; $t->height = 5;
$myCollection[] = $t; unset($t);
$c = new Circle; $c->radius = 3;
$myCollection[] = $c; unset($c);
$c = new Color; $c->name = "blue";
$myCollection[] = $c; unset($c);
foreach($myCollection as $s) {
   if($s instanceof Shape) {
      print("Area: " . $s->getArea() . "<br>},
   }
   if($s instanceof Polygon) {
      print("Sides: " . $s->getNumberOfSides() . "<br>\n");
   }
   if($s instanceof Color) {
      print("Color: $s->name<br>\n");
   }
   print("<br>}
   print("<br/>);
}
```

# Content

- 1. Creating an Object
- 2. Accessing attributes and methods
- 3. Building a class
- 4. Introspection

## 4. Introspection

- Ability of a program to examine an object's characteristics, such as its name, parent class (if any), attributes, and methods.
- Discover which methods or attributes are defined when you write your code at runtime, which makes it possible for you to write generic debuggers, serializers, profilers, etc

## 4.1. Examining Classes

- class exists(classname)
  - determine whether a class exists
- get declared classes()
  - returns an array of defined classes
- get\_class\_methods(classname)
  - Return an array of methods that exist in a class
- get\_class\_vars (classname)
  - Return an array of attributes that exist in a class
- get\_parent\_class(classname)
  - Return name of the parent class
  - Return FALSE if there is no parent class

```
function display_classes () {
    $classes = get_declared_classes();
    foreach($classes as $class) {
        echo "$howing information about $class<br />";
        echo "$class methods:<br />";
        $methods = get_class_methods($class);
        if(!count($methods)) {
            echo "<i>None</i>>        />";
        } else { foreach($methods as $method) {
                echo "<br/>        echo "<br/>        }
        echo "$class attributes:<br />";
        $attributes = get_class_vars($class);
        if(!count($attributes)) { echo "<i>None</i><br />"; }
        else {
            foreach(array_keys($attributes) as $attribute) {
                 echo "<br />";
            }
        }
        echo "<br />";
        }
        echo "<br />";
        }
    }
    echo "<br />";
}
```



## 4.2. Examining an Object

- is object(object)
  - Check if a variable is an object or not
- get class(object)
  - Return the class of the object
- method exists(object, method)
  - Check if a method exists in object or not
- get\_object\_vars( object)
  - Return an array of attributes that exist in a class
- get parent class(object)
  - Return the name of the parent class
  - Return FALSE if there is no parent class