



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Chapter 5. OOP

Object-Oriented Programming (OOP)

- **Object:** Instance (occurrence) 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

PHP 5

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

Content



1. Creating an Object

2. Accessing attributes and methods
3. Building a class
4. Introspection

1. Creating an Object

- Syntax:
 - *`$object = new Class([args]);`*
- E.g.:
 - `$obj1 = new User();`
 - `$obj2 = new User('Fred', "abc123"); //args`
 - `$obj3 = new 'User'; // does not work`
 - `$class = 'User'; $obj4 = new $class; //ok`

User
+ name - password - lastLogin
+ getLastLogin() + setPassword(pass)

Content

1. Creating an Object

→ 2. Accessing attributes and methods

3. Building a class

4. Introspection

2. Accessing Attributes and Methods

- *Syntax: Using ->*
 - *`$object->attribute_name`*
 - *`$object->method_name([arg, ...])`*
- E.g.

```
// attribute access
$obj1->name = "Micheal";
print("User name is " . $obj1->name);
$obj1->getLastLogin( ); // method call
// method call with args
$obj1->setPassword("Test4");
```

Content

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

3.1. Syntax to declare a Class

```
class ClassName [extends BaseClass]{  
    [[var] access $attribute [= value ]; ... ]  
    [access function method_name (args) {  
        // code  
    } ...  
]  
}
```

- access can be: **public**, **protected** or **private** (default is public).
- ClassNames, attributes, methods are case-sensitive and conform the rules for PHP identifiers
- attributes or methods can be declared as **static** or **const**

Rules for PHP Identifiers

- Must include:
 - ASCII letter (a-zA-Z)
 - Digits (0-9)
 - _
 - ASCII character between 0x7F (DEL) and 0xFF
- Do not start by a digit

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	Á	192	C0	Ł	224	E0	α
129	81	Ù	161	A1	Í	193	C1	ł	225	E1	β
130	82	É	162	A2	Ó	194	C2	Ť	226	E2	Γ
131	83	À	163	A3	Ú	195	C3	ţ	227	E3	π
132	84	Ä	164	A4	Ñ	196	C4	—	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	†	229	E5	σ
134	86	ä	166	A6	ª	198	C6	‡	230	E6	μ
135	87	ç	167	A7	º	199	C7	‡	231	E7	τ
136	88	ê	168	A8	¿	200	C8	Ł	232	E8	Φ
137	89	ë	169	A9	ƒ	201	C9	Ŕ	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	Ł	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	Ŧ	235	EB	Ø
140	8C	î	172	AC	¾	204	CC	‡	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	=	237	ED	∞
142	8E	Ä	174	AE	«	206	CE	‡	238	EE	ε
143	8F	Å	175	AF	»	207	CF	Ł	239	EF	Π
144	90	É	176	B0	☐	208	DO	Ł	240	FO	≡
145	91	æ	177	B1	☐	209	D1	Ŧ	241	F1	±
146	92	Æ	178	B2	☐	210	D2	Ŧ	242	F2	≥
147	93	ô	179	B3		211	D3	Ł	243	F3	≤
148	94	ö	180	B4	†	212	D4	Ł	244	F4	[
149	95	ò	181	B5	‡	213	D5	Ŕ	245	F5]
150	96	û	182	B6	‡	214	D6	Ŕ	246	F6	÷
151	97	ù	183	B7	ŋ	215	D7	‡	247	F7	≈
152	98	ÿ	184	B8	ŋ	216	D8	‡	248	F8	°
153	99	Ö	185	B9	‡	217	D9	Ŕ	249	F9	•
154	9A	Û	186	BA		218	DA	Ŕ	250	FA	·
155	9B	ó	187	BB	ŋ	219	DB	■	251	FB	√
156	9C	£	188	BC	ŋ	220	DC	■	252	FC	²
157	9D	¥	189	BD	ŋ	221	DD	■	253	FD	³
158	9E	ℳ	190	BE	ŋ	222	DE	■	254	FE	■
159	9F	f	191	BF	ŋ	223	DF	■	255	FF	□

Example – Define User class

```
//define class for tracking users
```

```
class User {
```

```
    public $name;
```

```
    private $password, $lastLogin;
```

```
    public function __construct($name, $password) {
```

```
        $this->name = $name;
```

```
        $this->password = $password;
```

```
        $this->lastLogin = time();
```

```
    }
```

```
    function getLastLogin() {
```

```
        return(date("M d Y", $this->lastLogin));
```

```
    }
```

```
}
```

User
+ name
- password
- lastLogin
+ getLastLogin()

A special variable
for the particular instance
of the class

3.2. Constructors and Destructors

- Constructor
 - **`__construct([args])`**
 - executed immediately upon creating an object from that class
- Destructor
 - **`__destruct()`**
 - calls when we want to destroy the object
- 2 special namespaces:
 - self: refers to the current class
 - parent: refers to the immediate ancestor
 - Call parents' constructor: **`parent::__construct`**

```
<?php
```

Example

```
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();
```

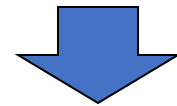
```
?>
```

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** or **Counter::\$count**
- Constant member
 - value cannot be changed
 - can be accessed directly through the class or within object methods using the **self** namespace.

Example

```
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>\n");  
$c2 = new Counter();  
print(Counter::getCount() . "<br>\n");
```

1

2

1

Version used: 2

```
$c2 = NULL;  
print($c1->getCount() . "<br>\n");  
print("Version used: ".Counter::VERSION."<br>\n");
```

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

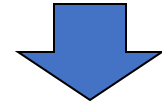
3.4. Object Cloning

- Special method in every class: **`__clone()`**
 - Every object has a default implementation for **`__clone()`**
 - Accepts no arguments
- Call cloning:
 - **`$copy_of_object = clone $object;`**
 - E.g.

```
$a = new SomeClass();  
$b = clone $a;
```

Example - Cloning

```
class ObjectTracker {  
    private static $nextSerial = 0;  
    private $id, $name;  
    function __construct($name) {  
        $this->name = $name;  
        this->id = ++self::$nextSerial;  
    }  
    function __clone() {  
        $this->name = "Clone of $this->name";  
        $this->id = ++self::$nextSerial;  
    }  
    function getId() { return($this->id); }  
    function getName() { return($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>");
```



Hello world!
1 Zeev's Object
2 Another object

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:

$$\$a = \$obj->b = 8;$$

Example - Attribute overloading

```

class PropertyTest {
    private $data = array();
    public $declared = 1;
    private $hidden = 2;
    public function __set($name, $value) {
        echo "Setting '$name' to '$value'<br>";
        $this->data[$name] = $value;
    }
    public function __get($name) {
        echo "Getting '$name'<br>";
        if (array_key_exists($name, $this->data)) {
            return $this->data[$name];
        }
    }
    public function __isset($name) {
        echo "Is '$name' set?<br>";
        return isset($this->data[$name]);
    }
    public function __unset($name) {
        echo "Unsetting '$name'<br>";
        unset($this->data[$name]);
    }
    public function getHidden() {
        return $this->hidden;
    }
}

```

Setting 'a' to '1'

Getting 'a'

1

Is 'a' set?

bool(true) Unsetting 'a'

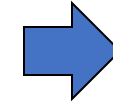
Is 'a' set?

bool(false)

1

2

Getting 'hidden'



```
$obj = new PropertyTest;
```

```
$obj->a = 1;
```

```
echo $obj->a."<br>";
```

```
var_dump(isset($obj->a));
```

```
unset($obj->a);
```

```
var_dump(isset($obj->a));
```

```
echo "<br>";
```

```
echo $obj->declared."<br>";
```

```
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.

Example – Method Overloading

```
class MethodTest {  
    public function __call($name, $arguments) {  
        // Note: value of $name is case sensitive.  
        echo "Calling object method '$name' "  
            . implode(', ', $arguments) . "<br>";  
    }  
  
    public static function __callStatic($name, $arguments) {  
        // Note: value of $name is case sensitive.  
        echo "Calling static method '$name' "  
            . implode(', ', $arguments) . "<br>";  
    }  
}
```

```
$obj = new MethodTest;
```

```
$obj->runTest('in object context');
```

```
MethodTest::runTest('in 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:

```
<?php
    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

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

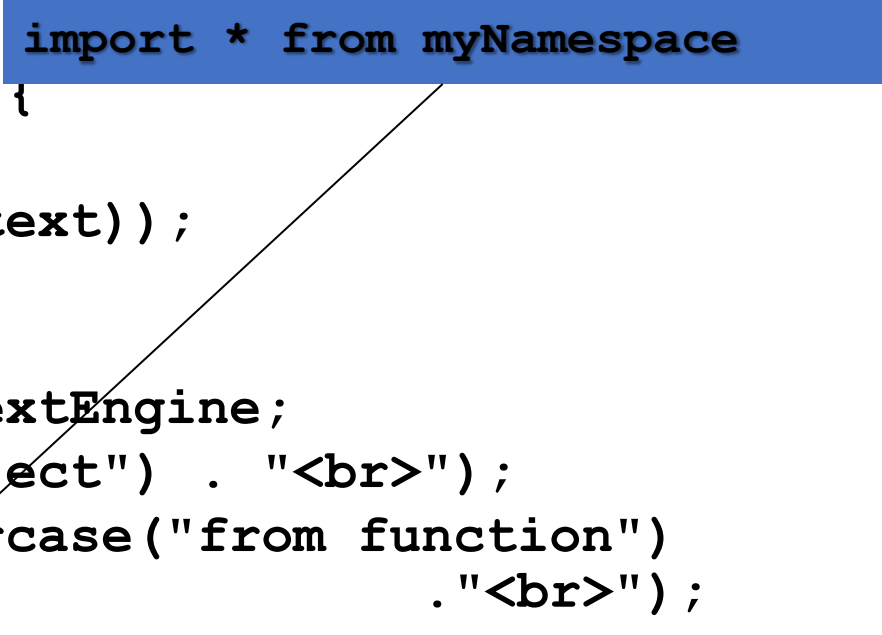
3.7. Namespace (2)

- 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:is1 { ... }**

Example - Namespace

```
namespace core_php:utility {  
    class TextEngine {  
        public function uppercase($text) {  
            return(strtoupper($text));  
        }  
    }  
}  
function uppercase($text) {  
    $e = new TextEngine;  
    return($e->uppercase($text));  
}  
}  
$e = new core_php:utility::TextEngine;  
print($e->uppercase("from object") . "<br>");  
print(core_php:utility::uppercase("from function")  
        . "<br>");  
  
import class TextEngine from core_php:utility;  
$e2 = new TextEngine;
```

`import * from myNamespace`



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

```
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) ;  
    }  
}
```

```
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: " . $s->getArea() . "<br>\n");
    }
    if($s instanceof Polygon) {
        print("Sides: " . $s->getNumberOfSides() . "<br>\n");
    }
    if($s instanceof Color) {
        print("Color: $s->name<br>\n");
    }
    print("<br>\n");
}

```

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 "Showing information about $class<br />";
        echo "$class methods:<br />";
        $methods = get_class_methods($class);
        if(!count($methods)) {
            echo "<i>None</i><br />";
        } else { foreach($methods as $method) {
                    echo "<b>$method</b>( )<br />";
                }
        }
        echo "$class attributes:<br />";
        $attributes = get_class_vars($class);
        if(!count($attributes)) { echo "<i>None</i><br />"; }
        else {
            foreach(array_keys($attributes) as $attribute) {
                echo "<b>\$$attribute</b><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

Question?

