IT4409: Web Technologies and e-Services

2020-1

Advanced PHP

Instructor: Dr. Thanh-Chung Dao
Slides by Dr. Binh Minh Nguyen

Department of Information Systems
School of Information and Communication Technology
Hanoi University of Science and Technology

1

Content

- Objects
- Defining objects
- Inheritance
- Sessions and session variables

2

Object oriented programming in PHP

- PHP, like most modern programming languages (C++, Java, Perl, JavaScript, etc.), supports the creation of objects.
- Creating an object requires you to first define an object class (containing variables and/or function definitions) and then using the "new" keyword to create an instance of the object class. (Note that the object must be defined before you instantiate it.)

```
<?php
//  Assume that the "Person" object has been previously defined. . .

$x = new Person;  //  creates an instance of the Person class (*no* quotes)

//  The object type need not be "hardcoded" into the declaration.

$object_type = 'Person';
$y = new $object_type;  //  equivalent to $y = new Person;

$z = new Vehicle('Jaguar','green');  //  creating an object and passing
                                     //      arguments to its constructor

?>
```

3

Defining (declaring) a class

- Use the "class" keyword which includes the class name (case-insensitive, but otherwise following the rules for PHP identifiers). Note: The name "stdClass" is reserved for use by the PHP interpreter.

```
<?php
class Person
{
   var $name;

    function set_name($new_name)  {
      $this->$name = $new_name;
   }

    function get_name() {
      return $this -> name;
   }


}
```

- Use the "$this" variable when accessing properties and functions of the current object. Inside a method this variable contains a reference to the object on which the method was called.

4

Declaring a class (cont.)

- Properties and functions can be declared as “public” (accessible outside the object’s scope), “private” (accessible only by methods within the same class), or “protected” (accessible only through the class methods and the class methods of classes inheriting from the class.
- Note that unless a property is going to be explicitly declared as public, private, or protected, it need not be declared before being used (like regular PHP variables).

```
<?php
class Person
{
   protected $name;
   protected $age;

   function set_name($new_name)  {
      $name = $this -> new_name;
   }

   function get_name() {
      return $this -> name;
   }

}
```

5

Declaring a class (cont.)

- Classes can also have their own constants defined (using the “const” keyword), can have their own static properties and functions (using the keyword “static” before “var” or “function”), and can also can constructors and destructors (see below).
- Static properties and functions are accessed (see below) using a different format than usual for objects, and static functions cannot access the objects properties (i.e. the variable $this is not defined inside of a static function).

```
<?php

class HTMLtable {
   static function start() {
     echo "<table> \n";
   }
   static function end() {
     echo "</table> \n";
   }
}

HTMLtable::start();

?>
```

6

Accessing properties and methods

- Once you have an object, you access methods and properties (variables) of the object using the –> notation.

```
<?php

$me = new Person;

$me -> set_name('Russ');
$me -> print_name();
$name = $me -> get_name();
echo $me -> get_name();

$age = 36;
$me -> set_age($age);

?>
```

7

Constructors and destructors

- Constructors are methods that are (generally) used to initialize the object's properties with values as the object is created. Declare a constructor function in an object by writing a function with the name __construct().

```
<?php
class Person {
   protected $name;
   protected $age;
      function __construct($new_name, $new_age)  {
          $this -> name = $new_name;
          $this -> age = $new_age;
      }
  // . . . other functions here . . .
}


$p = new Person('Bob Jones', 45);
$q = new Person('Hamilton Lincoln', 67);
?>
```

- Destructors (defined with a function name of __destructor()) are called when an object is destroyed, such as when the last reference to an object is removed or the end of the script is reached (the usefulness of destructors in PHP is limited, since, for example dynamic memory allocation isn't possible in the same way that it is in C/C++).

8

Inheritance

- Use the "extends" keyword in the class definition to define a new object that inherits from another.

```
<?php
  class Employee extends Person  {
     var $salary;

     function __construct($new_name, $new_age, $new_salary);  {
        $this -> salary = $new_salary;
        parent::__construct($new_name, $new_age);  //  call the constructor
                                                   //    of parent object
     }
     function update_salary($new_salary)  {
              $this -> salary = $new_salary;
     }
$emp = new Employee('Dave Underwood', 25, 25000);

?>
```

9

Inheritance (cont.)

- The constructor of the parent isn't called unless the child explicitly references it (as in this previous case). There is no automatic chain of calls to constructors in a sequence of objects defined through inheritance.
- You could "hard-code" the call to the parent constructor using the function call "`Person::__construct($new_name, $new_age);`" but it's typically better to define it in the manner given using the parent::method() notation. The same manner is used to call the method of a parent that has been overridden by its child.
- You can use the "self" keyword to ensure that a method is called on the current class (if a method might be subclassed), in this style `self::method();`
- To check if an object is of a particular class, you can use the `instanceof` operator.

```
if ($p instanceof Employee) {
    //  do something here
    }
```

10

More on classes

- You can also define interfaces for objects (for which any object that uses that interface must provide implementations of certain methods), and you can define abstract classes or methods (that must be overridden by its children).
- The keyword “final” can be used to denote a method that cannot be overridden by its children.

```
class Person {
   var $name;

   final function get_name() {
      return $this -> name;
   }
}
```

11

```php
<?php

// Declare the interface 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

// Implement the interface
// This will work
class Template implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value, $template);
        }

        return $template;
    }
}
```

12

```php
// This will not work
// Fatal error: Class BadTemplate contains 1 abstract methods
// and must therefore be declared abstract (iTemplate::getHtml)
class BadTemplate implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }
}
?>
```

13

More on classes (cont.)

- There are methods for "introspection" about classes, i.e. the ability of a program to examine an object's characteristics.

For example, the function `class_exists()` can be used (surprise!) to determine whether a class exists or not.

The function `get_declared_classes()` returns an array of declared classes.

```php
$classes = get_declared_classes();
```

You can also get an array of method names in any of the following (equivalent) manners:

```php
$methods = get_class_methods(Person);
$methods = get_class_methods('Person');
$class = 'Person';
$methods = get_class_methods($class);
```

14

More introspection functions

- There are a wide variety of introspection functions, several more are listed below.

```
get_class_vars($object);  /*  gets an associative array that maps
                              property names to values (including
                              inherited properties), but it *only*
                              gets properties that have default
                              values (those initialized with
                              simple constants)  */

is_object($object);  //  returns a boolean value

get_class($object);  /* returns the class to which an object
                        belongs  */

method_exists($object, $method);  // returns a boolean value

get_object_vars($object);  /*  returns an associative array
                               mapping properties to values (for
                               those values that are set (i.e.
                               not null)  */
```

15

A word about object methods…

- When defining the names of your own object methods, you should generally avoid starting them with a double underscore. Why?
- There are some "built-in" or predefined PHP method names that start in this manner, most notably constructors and destructors using the __construct() and __destructor() names. In the future (in new versions of PHP), it's possible that further methods might be defined that begin with a double underscore.
- Other reserved method names include __sleep() and __wakeup() which are used for object serialization and __get() and __set(), which can be defined so that if you try to access an object property that doesn't exist, these methods give an opportunity to either retrieve a value or set the (default?) value for that property. For example, if a class is used to represent data obtained from a database, you could write __get() and __set() methods that read and write data whenever requested.

16

PHP sessions

- By default, HTML and web servers don't keep track of information entered on a page when the client's browser opens another page. Thus, doing anything involving the same information across several pages can sometimes be difficult.
- *Sessions* help solve this problem by maintaining data during a user's visit, and can store data that can be accessed from page to page in your site.
- You can use session variables for storing information (this is one way that a "shopping cart" function can work for an online shop, for example).
- Servers keep track of users' sessions by using a session identifier, which is generated by the server when a session starts and is then used by the browser when it requests a page from the server. This session ID can be sent through a cookie (the default behavior) or by passing the session ID in the URL string.
- Sessions only store information temporarily, so if you need to preserve information, say, between visits to the same site, you should likely consider a method such as using a cookie or a database to store such information.

17

PHP sessions (cont.)

- To start a session, use the function session_start() at the beginning of your PHP script before you store or access any data. For the session to work properly, this function needs to execute before any other header calls or other output is sent to the browser.

```
<?php
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Session example</title>
</head>
<body>
<?php
include_once ('object.php');  //  Includes definition of the Person class
$_SESSION['hello'] = 'Hello world';
echo $_SESSION['hello'] . "<br/><br/>\n";
$_SESSION['one'] = 'one';
$_SESSION['two'] = 'two';
$me = new Person("Russ", 36, 2892700);
$_SESSION['name'] = $me->get_name();
echo "Testing " . $_SESSION['one'] ."," . $_SESSION['two'] . ", " . $me-
   >get_number() . " . . .<br/>\n";
?>
</body></html>
```

view the output page

18

Using session variables

- Once a session variable has been defined, you can access it from other pages.

```
<?php
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
<title>Session example 2</title>
</head>
<body>
<?php
echo "Welcome to a new page ". $_SESSION['name'] "!<br/>\n";
echo "Hope you enjoy your stay!  <br/>";
?>
<p>Back to regular HTML text...
</p>

</body>
</html>
```

view the output page

19

More on session variables

- You need to include a call to the session_start() function for each page on which you want to access the session variables.
- A session will end once you quit the browser (unless you've set appropriate cookies that will persist), or you can call the session_destroy() function. (Note, however, even after calling the session_destroy() function, session variables are still available to the rest of the currently executing PHP page.)
- The function session_unset() removes all session variables. If you want to remove one variable, use the unset($var) function call.
- The default timeout for session files is 24 minutes. It's possible to change this timeout.

20

Deleting all session variables

```
<?php
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
<title>Session example 3</title>
</head>
<body>
<?php
echo "Deleting all session variables using session_unset(); <br/>\n";
session_unset();
echo "Now the session variables are gone.  <br/>\n";
if (isset($_SESSION['name']))
   {  echo $_SESSION['name'] . "<br/>\n"; }
else
   {  echo "Session variable is not here.";  }
?>

</body>
</html>
```

view the output page

21

Learning Outcomes

- A (very, very brief) introduction to objects in PHP. If you wish to use objects, then many books on PHP include more information, and, of course, much information is available online.
- Sessions are useful for persistence of variables across many webpages without the need to submit information via forms.

22

email: chungdt@soict.hust.edu.vn

Q&A

23