CSCI 2910
Client/Server-Side Programming

Topic: More Topics in PHP

Reading: Williams & Lane pp. 108–121 and 232–243

Today's Goals

- Today we will begin with a discussion on objects in PHP including how to create instances and custom objects
- This will be followed by a discussion of PEAR along with some examples as to how the HTML_Template_IT package of PEAR can aid us with formatting.

Objects in PHP

- The concept of objects is the same across different object-oriented programming languages
- There are, however, minor differences between how a programmer references objects using PHP

Creating a New PHP Object Instance

- Just like JavaScript, PHP uses the keyword "new" to create a new instance of an object.
- Example: $_myinstance = new Object(args);
- Syntax elements:
  - Just like variables, the name used to identify the instance needs to begin with '$'.
  - Many objects need arguments (the "args" part of the above example) in order to create a new instance. These are passed to a function called a constructor which initializes the instance.

Referring to Components of a PHP Instance

- In JavaScript, we used periods to delimit/separate the elements of an object hierarchy. For example:

```
document.writeln("Hello, World!");
```

- In PHP, the operator "->" is used to delimit/separate the elements of an object hierarchy. For example:

```
$object_name->object_function();
```

- As the parenthesis indicate, the above refers to a function. The same format is used for properties too, i.e., `$object_name->property;`

Defining a Class

- A class is the definition used to create an instance.
- A class definition defines the class's name, its variables, and functions.
- A class definition can also contain functions used to initialize instances (constructors) and remove them (destructors).

Format of a Class Definition

```
<?php
// Basic format of a class definition

class ClassName
{
// Member variables
   var $_variable1 = 0;
   var $_variable2 = "String";

// Member functions
   function classFunction($_arg1 = 0, $_arg2)
   {
        // Function code goes here
   }
?>
```

Format of a Class Definition (continued)

- The keyword "class" followed by the class name is used to start the definition. Curly brackets are used to enclose all of the elements of the defintion.
- The keyword "var" is used to identify the class' variables.
- Variables can be initialized. Every time a new instance is created, the variables for that instance are initialized to these values.
- Functions are defined normally, but when contained within the curly brackets of the class, become member functions of the class.

Private Member Variables

- There are some cases when a class may not want to have its variables accessible outside of the class
  - Variables may be set up only for internal use within the class' functions
  - Variables may have certain restrictions on values that must be enforced internally
- If a variable needs to be modified from outside the class, a function can be provided to do so. For example, instead of:

```
        $_instance -> variable1 = 25;
```

  use

```
     $_instance -> updateVariable1(25);
```

Private Member Variables

- To declare a variable as private, simply replace the keyword "var" with the keyword "private" in the variable declaration.
- Example:
  `private $_variable3 = 4.0;`
- A class can also have private member functions. In this case, declare the function by putting the keyword "private" in front of the function declaration.
- Private variables are only available in PHP 5.

Static Member Variables

- Each time an instance of a class is created, a whole new set of variables and functions for that instance is created along with it.
- It is possible to make it so that regardless of the number of instances of a class, only a single variable is created for that class.
- This allows all instances to share a single variable.
- To do this, replace the keyword "var" with the keyword "static" in the variable declaration.
- Static variables are only available in PHP 5.

Constructors

- When an instance is created, it may be necessary to go through an initialization process.
- This initialization process might be based on arguments passed from the code creating the instance.
- A function can be written for a class that is automatically called whenever an instance for that class is created. This is called a constructor.

Constructors (continued)

- A constructor has the same format as a regular function except for the name.
- The name of a constructor in PHP 5 is _ _construct(). In PHP 4 it has the same name as the class.
- Example:

```
function _ _construct($_arg = 0)
{
  // Code to initialize class
}
```

- Note: I have put a space between the underscores to show there are two of them. No space is used.

Destructors

- It is also possible that some housekeeping or cleanup needs to be performed when an instance is removed.
- In this case, a destructor function is automatically called to close the instance.
- Destructors are only available in PHP 5.
- Unlike the constructor function, no arguments can be passed to the destructor function.
- The name of a destructor is always _ _destruct().

Class Definition Example

```
class Person
{
    var $full_name;
    var $birthday;
    var $gender;
// Print function
    function printPersonInHTML()
    {
        print "<p>{$this->full_name} is a ";
        if(($this->gender == 'M')||($this->gender == 'm'))
            print "male";
        else
            print "female";
        print
            " who was born on {$this->birthday}.</p>";
    }
```

Class Definition Example (continued)

```
// Constructor
    function Person($first_name, $last_name,
                    $gender, $birth_month,
                    $birth_day, $birth_year)
    {
        $month_list = array ("January", "February",
                      "March", "April", "May", "June",
                      "July", "August", "September",
                      "October", "November", "December");
        $this->full_name = $first_name." ".$last_name;
        $this->birthday =
            $month_list[$birth_month-1]." ".
            $birth_day.", ". $birth_year;
        $this->gender = $gender;
    }
}
```

Class Definition Example (continued)

- The code to create an instance and call the class function printPersonInHTML() looks like this:

```
$person_1 = new Person("John", "Doe",
   "m", 3, 24, 1974);

$person_1 -> printPersonInHTML();
```

- The output then will be:

```
John Doe is a male who was born on
March 24, 1974.
```

PHP Predefined Objects

- As with other languages, PHP has a number of predefined objects and functions that provide access to system resources.
- One package containing these objects and functions is called the PHP Extension and Application Repository or PEAR.
- It includes support for:
  - •Web services
  - •Image processing
  - •File handling
  - •Data validation
  - •Database access
  - •Payment processing
- PEAR was originally designed to support scripting for HTML such as providing templates for documents and platform independence.

PEAR Overview

The following descriptions of PEAR are copied from the pear.php.net website (source: http://pear.php.net/manual/en/introduction.php):

– "A structured library of open-sourced code for PHP users"
– "A system for code distribution and package maintenance"
– "A standard style for code written in PHP"
– "The PHP Extension Community Library (PECL)"
– "A web site, mailing lists and download mirrors to support the PHP/PEAR community"

PEAR Components

- First, we need to make sure the server you're using has PEAR installed and see which packages it has.
- At the Unix command prompt, type "pear list". The output shown below is from Einstein:

```
Installed packages:
===================
Package              Version State
Archive_Tar          1.1     stable
Console_Getopt       1.2     stable
HTML_Template_IT     1.1     stable
Net_UserAgent_Detect 2.0.1   stable
PEAR                 1.3.5   stable
XML_RPC              1.2.2   stable
```

- We will be using the HTML_Template_IT package

Using HTML Templates

- Throughout this course, templates have been presented to offer a starting point for your web page development.
- Templates simplify the development process by allowing the programmer to avoid the tedious stuff.
- PEAR allows programmers to separate the HTML code from the PHP scripts.
- The PEAR package HTML_Template_IT allows us to do just that.

Using HTML_Template_IT

- First of all, the use of templates requires two files:
  – an HTML template with placeholders for values
  – PHP code to insert values at the placeholders
- The HTML template looks just like a normal HTML file except that there are additional tags to show where the PHP script is to insert values.
- The PHP script determines the values that are to be inserted into the HTML template at execution time, and the resulting HTML output is sent to the client.

HTML_Template_IT Blocks

- The HTML template is divided into regions called blocks.
- These blocks are used by PHP to identify the region being processed.
- The format of a block is

```
<!-- BEGIN block_name -->
... block content ...
<!-- END block_name -->
```

- The name of a block can consist of upper and lowercase letters, underscores and hyphens. There can be no spaces in a block name.

HTML_Template_IT Placeholders

- Placeholders are located within a block of the HTML template to identify positions where the PHP script will insert values
- The format of placeholder is `{placeholder_name}`
- The placeholder name can consist of upper and lowercase letters, underscores and hyphens.
- The placeholder name must be placed between curly brackets without any spaces.
- Examples:

```
{page_title}
{menuitem-1}
```

Sample HTML Template

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

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

<head>
<title>Simple XHTML Document</title>
</head>
<body>
   <!-- BEGIN TITLE_BLOCK -->
        <h1>{page_title}</h1>
        <p>{page_intro}</p>
   <!-- END TITLE_BLOCK -->
</body>
</html>
```

Populating the Template

- PHP is then used to populate the template
- Associating a PHP script with an HTML template involves seven steps:
  1. Include the PEAR Integrated Template
  2. Create a template object to be used by the PHP script for function calls
  3. Associate the template file with the object
  4. Select a block to work with
  5. Assign data to the placeholders
  6. Parse (process) the block
  7. Output the page

Including the PEAR IT

- Including the PEAR Integrated Template is the same as including any file. It is recommended that you use the require_once() function.
- require_once() includes the specified file exactly once during the execution of the script, i.e., it prevents multiple includes.
- The file to include is IT.php which may appear in different places on different servers.
- Einstein has IT.php in the folder "/usr/local/lib/php/HTML/Template/"
- Code example:
  ```
  require_once
  ("/usr/local/lib/php/HTML/Template/IT.php");
  ```

Creating the Template Object

- Creating the template object is the same as creating any object using a constructor function.
- Code example:
  ```
  $template = new
  HTML_Template_IT("./template_folder");
  ```
- The argument for the constructor function is the directory where the templates will be found.
- The "./" points to the current folder while "template_folder" identifies a sub-folder.

Associate the Template File

- Now we need to associate a template file with the template object. This is done with the HTML_Template_IT function *loadTemplatefile().*
- Code example:
  ```
  $template->
  loadTemplatefile("template_01.tpl",
  true, true);
  ```
- The first argument is the template file name
- The second and third arguments tell the script how to handle undefined blocks and placeholders.

Selecting a Block

- Since there may be multiple blocks within the template, the PHP script must identify which block is being used.
- This is done with the HTML_Template_IT function *setCurrentBlock()*.
- Code example:
  ```
  $template->
  setCurrentBlock("TITLE_BLOCK");
  ```

Assign Data to the Placeholders

- Once a block is selected, the placeholders need to be populated.
- This is done using the HTML_Template_IT function *setVariable()*.
- Code example:
```
$template->setVariable("page_title",
"Hello, World!");
```

Parsing/Processing the Block

- Once you are finished setting the values of a block, it can be parsed or processed.
- This is done using the HTML_Template_IT function *parseCurrentBlock()*.
- Code example:
```
$template->parseCurrentBlock();
```

Outputting the Page

- After you have finished processing all of the blocks, the page must be output.
- This is done using the HTML_Template_IT function *show()*.
- Code example:
```
$template->show();
```

Final PHP Script Using Templates

```
<?php
// Load PEAR's Integrated Template class into the script
    require_once ("/usr/local/lib/php/HTML/Template/IT.php");
// Create a new template, and specify that the template files are in the subdirectory
    "template_folder"
    $template = new HTML_Template_IT("./template_folder");
// Load the necessary template file
    $template->loadTemplatefile("template_01.tpl", true, true);
// Identify which block of the template we're working with
    $template->setCurrentBlock("TITLE_BLOCK");
// Assign the data values to the template placeholders
    $template->setVariable("page_title", "Hello, World!");
    $template->setVariable("page_intro", "Our first PHP script using HTML templates!");
// Process the current block
    $template->parseCurrentBlock();
// Output the web page
    $template->show();
?>
```

The Result

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

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

<head>

<title>Simple XHTML Document</title>
</head>
<body>
       <h1>Hello, World!</h1>
       <p>Our first PHP script using HTML templates!</p>
</body>
</html>
```

Loops with Templates

- By parsing the blocks properly, a loop can be used to generate HTML code.
- For example, we can use a loop to generate successive rows of a table.
- The process would be something like this:
  – Print the start tag for the table
  – Begin a block
  – Print a row with placeholders for the PHP values
  – End the block
  – Print the end tag for the table
- Executing the block multiple times will create multiple rows

Loops with Templates (continued)

```
<body>
  <table align="center" border="2"
  cellpadding="5">
  <!-- BEGIN TABLE_HEADING -->
     <tr><td>{column1}</td>
     <td>{column2}</td></tr>
  <!-- END TABLE_HEADING -->
  <!-- BEGIN TABLE_BLOCK -->
     <tr><td>{column1}</td>
     <td>{column2}</td></tr>
  <!-- END TABLE_BLOCK -->
  </table>
</body>
```

Loops with Templates (continued)

- As far as using this template with a PHP script is concerned, the PHP script will need to insert the values into the placeholders once for each execution of the loop
- The process inside the PHP loop would be something like this:
  - Set the current block
  - Set the values for the different placeholders
  - Parse the current block
- Each time the loop was executed, a new row would be created.

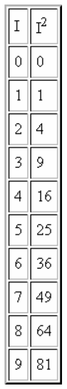
Loops with Templates (continued)

```
<?php
   require_once ("/usr/local/lib/php/HTML/Template/IT.php");
   $template = new HTML_Template_IT("./template_folder");
   $template->loadTemplatefile("template_02.tpl", true, true);
// Create table column headings
   $template->setCurrentBlock("TABLE_HEADING");
   $template->setVariable("column1", "I");
   $template->setVariable("column2", "I<sup>2</sup>");
   $template->parseCurrentBlock();

// Create the 10 rows one at a time
   for ($i = 0; $i <10; $i++)
   {
        $template->setCurrentBlock("TABLE_BLOCK");
        $template->setVariable("column1", $i);
        $template->setVariable("column2", ($i*$i));
        $template->parseCurrentBlock();
   }
   $template->show();
?>
```

Result

Okay, so it isn't a beautiful example, but it is a beginning. Imagine how much we could help the output of the database query outputs using this sort of tool.



I	I^2
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81

Printing MySQL Query Results with Templates

If we replace the code from the earlier example with the results from fetching each record from a MySQL query, we could significantly improve the format of the output.

```
<body>
  <table align="center" border="0" cellpadding="5">
    <!-- BEGIN TABLE_BLOCK -->
          <tr>
          <td>{column1}</td>
          <td>{column2}</td>
          <td>{column3}</td>
          <td>{column4}</td>
          </tr>
    <!-- END TABLE_BLOCK -->
  </table>
</body>
```

Outputting PHP MySQL Results

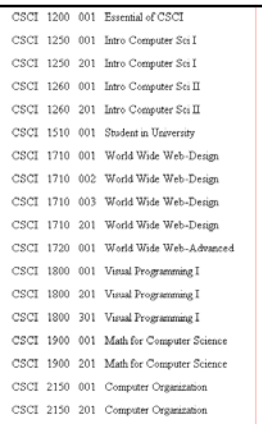
```
<?php
// First, connect to the template we're going to use
   require_once ("/usr/local/lib/php/HTML/Template/IT.php");
   $template = new HTML_Template_IT("./template_folder");
   $template->loadTemplatefile("template_03.tpl", true, true);

// Next, get the result of a database query
   $c = mysql_connect ("localhost", "zxyx999", "12345");
   mysql_select_db("zxyx999", $c);
   $result = mysql_query("SELECT DEPT, COURSE, SECTION, TITLE from timetable", $c);

// Go through the records prin
   while($record = mysql_fetch_array($result, MYSQL_ASSOC))
   {
        $template->setCurrentBlock("TABLE_BLOCK");
        $template->setVariable("column1", $record[DEPT]);
        $template->setVariable("column2", $record[COURSE]);
        $template->setVariable("column3", $record[SECTION]);
        $template->setVariable("column4", $record[TITLE]);
        $template->parseCurrentBlock();
   }
   mysql_close ($c);
   $template->show();
?>
```

Result

- This makes formatting a great deal easier. In addition, a single template can serve multiple PHP scripts.



CSCI	1200	001	Essential of CSCI
CSCI	1250	001	Intro Computer Sci I
CSCI	1250	201	Intro Computer Sci I
CSCI	1260	001	Intro Computer Sci II
CSCI	1260	201	Intro Computer Sci II
CSCI	1510	001	Student in University
CSCI	1710	001	World Wide Web-Design
CSCI	1710	002	World Wide Web-Design
CSCI	1710	003	World Wide Web-Design
CSCI	1710	201	World Wide Web-Design
CSCI	1720	001	World Wide Web-Advanced
CSCI	1800	001	Visual Programming I
CSCI	1800	201	Visual Programming I
CSCI	1800	301	Visual Programming I
CSCI	1900	001	Math for Computer Science
CSCI	1900	201	Math for Computer Science
CSCI	2150	001	Computer Organization
CSCI	2150	201	Computer Organization

More on loadTemplatefile()

- We haven't yet discussed the last two arguments of loadTemplatefile().
- Our code example was
```
$template->
loadTemplatefile("template_01.tpl", true,
true);
```
- The first argument identifies the template file.
- The second argument is set to "true" if you want the PHP engine to ***not*** print out blocks from the template that were not used in the script.
- The third argument is set to "true" if you want the PHP engine to ***not*** print out placeholders that have not had values assigned to them.