Coding Standards for CMSC 128
OpenLGU Project

I. FILE FORMATTING

A. PHP-only files

Files that contain PHP code ONLY should no longer include the ?> delimiter to prevent the inclusion of accidental injection of empty lines after the response.

B. Indentation

Do NOT use tab spaces (‘\t’). Substitute four (4) spaces for tabs.

C. Maximum Line Length

Try to keep each line of PHP code to 80 columns. If necessary, the absolute maximum line length is 120. To check the length of your line, look for the Column (Col) statistic in the status bar of your text editor.

D. Line Termination

Since we are going to deploy the system in a Linux/Unix environment, use the Unix text file convention. Lines must end

with a single linefeed (LF; ‘\n’). If you are editing your PHP file in Notepad++, you can change this by going to Edit > EOL Conversion > UNIX Format. This is different from line termination in Windows (CRLF; ‘\r’’\n’), and Mac (CR; ‘\r’).

II. NAMING CONVENTIONS

Remember to always make the names of your classes fit. Ensure that classes are encapsulated in **namespaces**, a mechanism that is now supported in PHP 5.3.0 onwards (this is similar in application to packages in Java).

A. Classes

Class names may only contain alphanumeric characters, and should accurately describe the class it is named for. Class names’ first letter should be capitalized. Should the name contain more than one word, each word’s first letter must be capitalized. However, consecutive capitalized letters are not allowed; abbreviations such as HTML and

PHP should instead by stylized as Html and Php. Examples of valid class names are: ExecutiveManager, PdfViewer.

B. File Names

File names should only contain alphanumeric characters, underscores, and dash (‘-‘) characters. NO SPACES ARE PERMITTED. Although permitted, avoid underscores and dash characters as much as possible. If a PHP file contains a class, its file name should map to the class’s name (including capitalization). Otherwise, PHP file names may start with lower case alphabetic characters and follow camel case (each succeeding word has its first letter capitalized). The extension for PHP files is .php (in lower case). For example, file names such as index.php, profile.php, userIndex.php are permitted as long as they don’t contain classes. A PHP file containing a class ExecutiveManager should be named ExecutiveManager.php.

C. Functions and Methods

Functions and methods may only contain alphanumeric characters although numbers are discouraged. Underscores are NOT permitted. Function names must always start with a lower case letter and follow camel case. Try to be as verbose as possible in using a function's name to describe what it does. However, be practical; using an overly long function name will require you to type it over and over again to use the function.

Accessors (get and set) methods should always be prefixed by "get" or "set" as in Java.

D. Variables

Variable names may only contain alphanumeric characters although numbers are discouraged, as in function names. Underscores are NOT permitted. Variable names should start with a lower case letter and follow camel case.

E. Constants

Constants may contain alphanumeric characters and underscores. All letters must be capitalized and words must be separated by an underscore. Constants are defined using the const modifier. This is the only way a variable can be considered a constant and should follow this rule.

III. CODING STYLE

A. PHP Code Delimiters

ALWAYS use <?php as the opening delimiter and ?> as the closing delimiter (when necessary; refer to I-A).

B. Strings

String literals (strings with no variable substitutions) should be delimited by single quotes (apostrophes), for example:

```
$a = 'Example String'
```

If a string literal itself contains apostrophes, use double quotes to delimit them instead. This is particularly applied to SQL statements, for example:

```
$sql = “SELECT ‘id’, ‘name’
from ‘people’ “
        . “WHERE ‘name’=‘Fred’
OR ‘name’=‘Susan’”;
```

This is better than escaping apostrophes because it is much easier to read.

To substitute variables in strings, delimit the string with double quotes (“”) and enclose the variable with curly braces ({})

```
$greeting = “Hello {$name},
welcome back!”
```

Strings may be concatenated using the . operator. Its use is encouraged in order to break down a long statement/string into multiple lines, in which case, the . must be aligned with the = symbol. An example has already been given above.

C. Arrays

Always use 0 as the base index. Negative indices are not allowed.

When declaring arrays using the array function, a trailing space must be entered after each comma delimiter.

```
$sampleArray = array(1, 2, 3, 'Zend', 'Studio');
```

Array declarations may be multiline, but each successive line must be aligned as follows:

```
$sampleArray = array(1, 2, 3, 'Zend', 'Studio',
                     $a, $b, $c,56.44, $d, 500);
```

D. Class Declarations

Remember the class naming standard described in II-A. All parent superclasses and implemented interfaces must be declared on the same line as the class name. If the line exceeds the maximum line length limit, break the line at the extends keyword and at the implements keyword. The opening brace for a class should always be written on the line underneath

the class name. All code inside the class declaration must be indented (first level). Only one class is permitted per PHP file. If a PHP file is a class PHP file, no other PHP code must be contained therein. Finally, the closing brace for the class declaration should be one the line after the last line of code. For example:

```
class SampleClass
{
//member variables
//funtions
//code here
}
```

E. Class Member Variables

Member variables must also follow the naming convention stated in II-D. They should be declared at the top of the class definition, before any method declarations. Variable declarations should *not* use the var construct. They should always declare visibility via the keywords private, protected, or public. Public member variables are discouraged in favor of private variables

accessed through get and set methods.

F. Function and Method Declarations

Class functions and methods should still follow the naming convention stated in II-C. As with class member variables, they should also always declare their visibility via the keywords private, protected, or public. As with classes, the opening brace should be placed one line underneath the class name. There should be no spaces in between the function name and the opening parentheses for arguments. There should be a space after each comma delimiting each argument in the list. Global functions are strongly discouraged.

If argument lists are too long and exceed the maximum line length limit, a line break should be introduced after the comma of the last argument that fits in the initial line, and the next line containing the arguments should be aligned with the first argument in the list. The opening parentheses of the function body should be one line underneath the last line of the argument list.

Pass-by-reference is the only parameter passing mechanism permitted in a method declaration. Moreover, call-time pass-by-reference is strictly prohibited.

Return values must be enclosed in parentheses to maintain readability.

When passing variables to a function, they should there should be one space after each comma delimiting each argument.

G. Control Statements

Operators in conditionals should always be delimited by white spaces. If conditions are too long, introduce line breaks at the end of each simple expression. The succeeding lines should be aligned with the first simple expression. Any content within the body of a control statement should be indented with four (4) spaces.

For switch statements, the default case should never be omitted.

All control statements must use braces.

H. Inline Documentation

phpDocumentor will be used to generate the documentation for our files. Each PHP file should have a docblock. If the file contains a class, the class should also have a docblock of its own, and each of the class's methods should have a docblock of their own. Each docblock should contain a short description and long description of the file/class/function that it is describing.

At the file level, the following phpDoc tags are required:

- @category
- @package
- @subpackage
- @versio0n
- @since
- @author

At the class level, the following phpDoc tags are required:

- @category
- @package
- @subpackage
- @version

- @since
- @author

At the function level, the following phpDoc tags are required:

- @param
- @return
- @throws (only if the function may throw an exception)
- @author

The general format for a docblock will then be:

```
/**
 * Short description
 *
 * Long description (if any)
 *
 * <phpDoc tags (in the order specified above)>
 */
```

For more information about the tags specified above, go to: http://goo.gl/62PbT.