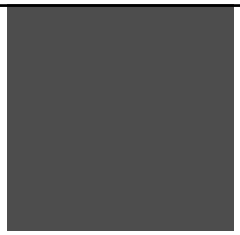


Web Application Development: Debugging, Patterns, Regular Expressions and Object-oriented PHP

Week 10

SWINBURNE UNIVERSITY OF TECHNOLOGY

Content

- Debugging and Error Handling
- Patterns
- Regular Expressions
- Object-Oriented Programming
- Using Classes in PHP scripts
- Defining PHP Classes

2

Three Types of Errors (Bugs)

- **Syntax (or parse) Errors** occur when the scripting engine fails to recognize code. Syntax errors can be caused by incorrect use of code, references to objects, methods, and variables that do not exist, or mistyped words
  - Syntax errors in compiled languages are also called **compile-time errors**
  - In PHP, Syntax Errors generate *Parse error messages*
- **Run-Time Errors** occur when the scripting engine encounters a problem while a program is executing. Run-time errors do not necessarily represent language errors. Run-time errors occur when the scripting engine encounters code that it cannot execute
  - In PHP, depending on severity, Run-Time Errors generate either *Fatal error messages*, or *Warning messages*, or *Notice messages*
    - **Logic Errors** are flaws in a program's design that prevent the program from running as anticipated
  - Logic Errors do not generate error messages ☹

3

Four Types of Error Messages in PHP

- **Parse error messages** occur when a script contains a syntax error that prevents your script from running.
- **Fatal error messages** are raised when a script contains a run-time error that prevents it from executing, e.g., call a wrong function.
- **Warning messages** are raised for run-time errors that do not prevent a script from executing, e.g., pass the wrong number of parameters to a function.
- **Notice messages** are raised for potential run-time errors that do not prevent a script from executing, e.g., attempt to use an undeclared variable.

http://php.net/manual/en/book.errorfunc.php

4

Printing Errors to the Web Browser

- The php.ini configuration file contains two directives that determine whether error messages print to a Web browser:
  - `display_errors` directive prints script error messages and is assigned a value of "On"
  - `display_startup_errors` directive displays errors that occur when PHP first starts and is assigned a value of "Off"
- The `error_reporting` directive in the php.ini configuration file determines which types of error messages PHP should generate. By default, the `error_reporting` directive is assigned a value of "E_ALL," which generates all errors, warnings, and notices to the Web browser
- Alternatively, place the function at the beginning of a script section.
  - `error_reporting(E_ALL &~ E_NOTICE);`
  - `error_reporting(E_ERROR | E_PARSE);`

5

Setting the Error Reporting Level

Constant	Integer	Description
--	0	Turns off all error reporting
E_ERROR	1	Reports fatal run-time errors
E_WARNING	2	Reports run-time warnings
E_PARSE	4	Reports syntax errors
E_NOTICE	8	Reports run-time notices
E_CORE_ERROR	16	Reports fatal errors that occur when PHP first starts
E_CORE_WARNING	32	Reports warnings that occur when PHP first starts
E_COMPILE_WARNING	32	Reports warnings generated by the Zend Scripting Engine
E_COMPILE_ERROR	64	Reports errors generated by the Zend Scripting Engine
E_USER_ERROR	256	Reports user-generated error messages
E_USER_WARNING	512	Reports user-generated warnings
E_USER_NOTICE	1024	Reports user-generated notices
E_ALL	2047	Reports errors, warnings, and notices with the exception of E_STRICT notices
E_STRICT	2048	Reports strict notices, which are code recommendations that ensure compatibility with PHP 5

6

Writing Custom Error-Handling Functions

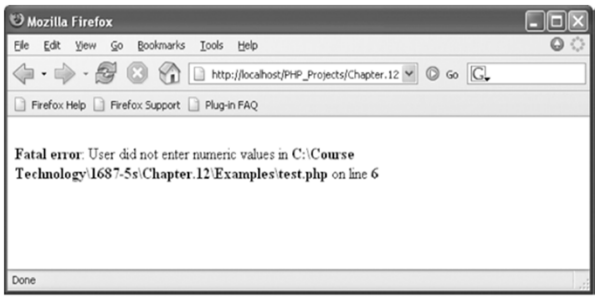
- Use the `set_error_handler()` function to specify a custom function to handle errors
  - Suppose `processErrors()` is a custom error-handling function, then `set_error_handler("processErrors")`
- Custom error-handling functions can only handle the following types of error reporting levels:
  - `E_WARNING, E_NOTICE, E_USER_ERROR, E_USER_WARNING, E_USER_NOTICE.`
- All other types of error reporting levels are handled by PHP's built-in error-handling functionality.
- Once you use `set_error_handler()`, none of PHP's default error-handling functionality executes for the preceding types of error reporting levels.
- To print the error message to the screen, you must include `echo()` statements in the custom error-handling function

7

The `trigger_error()` Function

- Use the `trigger_error()` function to generate an error in your scripts. The function accepts two parameters: an error message and an error reporting level

```
if (isset($_GET['height']) && isset($_GET['weight'])) {
  if (!is_numeric($_GET['weight'])||!is_numeric($_GET['height'])){
    trigger_error("User did not enter numeric values", E_USER_ERROR);
    exit();
  }
} else trigger_error("User did not enter values", E_USER_ERROR);
$bodyMass = $_GET['weight']/($_GET['height']*$_GET['height'])*703;
printf("<p>Body mass index is %d.</p>", $bodyMass);
```



8

Tracing Errors with **echo( )** and Comments

- Tracing is the examination of individual statements in an executing program
- The `echo( )` statement provides one of the most useful ways to trace PHP code
- Place an `echo( )` method at different points in your program and use it to display the contents of a variable, an array, or the value returned from a function
- Another method of locating bugs in a PHP program is to "comment out" problematic lines
- The cause of an error in a particular statement is often the result of an error in a preceding line of code

9

Analysing Logic

- Errors from Logic problems are difficult to spot using tracing techniques.
- When you suspect that your code contains logic errors, you must analyse each statement on a case-by-case basis

```
// missing braces
if (!isset($_GET['firstName']))
      echo "<p>You must enter your first name!</p>";
      exit();
echo "<p>Welcome to my Web site, " . $_GET['firstName'] ."!";

// unexpected ";"
for ($count = 1; $count < 6; $count++);
       echo "$count<br />";
```

10

JavaScript Error Handling

Name	Description
EvalError	Raised when the eval() functions is used in an incorrect manner
RangeError	Raised when a numeric variable exceeds its allowed range
ReferenceError	Raised when an invalid reference is used
SyntaxError	Raised when a syntax error occurs while parsing JavaScript code
TypeError	Raised when the type of a variable is not as expected
URIError	Raised when the encodeURI() or decodeURI() functions are used in an incorrect manner

```
try {
  var x;
  x[1] = 3;
}
catch (err) {
  alert ('error: ' +
  err.message);
}
finally {
  alert ('complete');
}
var y = x;
```

Execute the code in **blue**.

If no errors, proceed to finalisation code in **green**, and then proceed to code in **black**.

If an error in blue code, execute the code in **red**. Then proceed to the finalisation block in **green**, and then the code in **black**.

Testing Code

```
<!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">
   <head>
      <script type="text/javascript" src="exception.js"></script>
       Testing Exception Handling
   </head>
   <body onLoad="exceptionTest()">
   </body>
</html>
```

Exception.htm

```
exceptionTest = function() {
   alert('starting');
   try {
      var x;  x[1] = 3;
   }
   catch (err) {
      alert ('error: ' + err.message);
   }
   finally {
      alert ('complete');
   }
}
```

exception.js

12

Try / Catch / Finally

- Enclose a block of code between try and catch; this is called the try block.
- catch is followed also by a block of code – the catch block
- This can be optionally followed by a finally block
- If an error occurs in the try block, control jumps straight to the catch block, which contains the code to execute once things have gone wrong (including usually alerting the user)
- If there is a finally block, this executes after either the try block or the catch block has completed
- In the catch block, details of the error that occurred are available – note the catch(err) in the code; err is a global variable, set when the error occurs, and this is passed in (like a parameter) to the code in the catch block. This object err can be accessed in the block; in our example its message property is printed out, to give some indication as to the kind of error that occurred.

13

The onerror Event Handler

```
<!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">
<head><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Accessing the error event</title>
<script type="text/javascript">
function handleError (msg, url, line) {
  alert ('An error occurred:\nmessage: ' + msg + '\nurl: ' + url +'\nline: ' + line);
  return true;
}
</script>
</head>
<body>
   <script type="text/javascript">
      onerror = handleError;
      var x;
      x[3] = 3;
   </script>
</body>
</html>
```

Error.htm

14

Error Message

onerror

- Here we make use of the fact that when there is a JavaScript error, the onerror event fires
- We set an event handler for this event, within the script whose error we wish to handle
- We write the code for this handler that will do what we wish when an error occurs
- In the previous slide we put both the erroneous JavaScript and the error handling script into the HTML file. Often both would be in external files.

16

Other Debugging Tools

- Mozilla JavaScript Console
- Microsoft Script Debugger – debugIt()
- Firefox's Firebug

17

DOM Inspectors

- Used to check a dynamic model of the DOM
- Traverses the browser's current DOM model and displays the current state of the DOM
- Displays the information of the selected elements
  - the values for properties
  - the CSS rules and styles
  - All properties of the JavaScript object that represents the selected element
- DOM Inspectors
  - Firefox DOM Inspector
  - IE DOM Inspector
  - Mouseover DOM Inspector (for Firefox, Mozilla, Opera7.5, Netscape 7, IE 6)

18

AJAX 'Patterns'

- Technology specific - not strict design patterns
- http://ajaxpatterns.org/Patterns#Ajax_Architecture
  - Programming patterns
  - Functionality and usability patterns
  - Development practices – diagnosis and testing
- Some examples from Ullman Dykes AJAX text
  - Form validation
  - Refresh
  - Handling errors

19

Form Validation Pattern

- Reliable programs need server-side as well as client – side validation
  - Javascript validation typically validates user input from the client
  - Conventionally server-side validation requires a form submit
- AJAX allows dynamic server-side validation
  - Triggered by user event (e.g. lost focus)
  - Or triggered periodically

20

FormValidation.htm

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
   <title>Form Validation Example</title>
   <script type="text/javascript" src="FormValidation2.js"></script>
</head>
  <body>
      <form name="form1" id="form1" method="post" action="formcheck.aspx">
         Preferred User Name: <input id="UserName" type="text"
         onblur="Validate('UserName')"  />
         …

      </form>
  </body>
</html>
```

Javascript function that calls uses AJAX to call server database to check if name is already taken

21

FormValidation.js

```
function Validate(data){
   var bodyofform = getBody(data);
   if (bodyofform != "UserName=")    {
      xHRObject.open("POST", "Validate.php", true);
      xHRObject.setRequestHeader
        ("Content-Type", "application/x-www-form-urlencoded");
      xHRObject.onreadystatechange = getData;
      xHRObject.send(bodyofform); ;
   }
   else  {
      span.innerHTML = "Blank user names not allowed";
   }
}
function getBody(data){
    var argument = data + "=";
    argument += encodeURIComponent(document.getElementById(data).value)
    return argument;
}
```

'UserName'

22

Polling the Server Pattern

- Continually polls the server to see if information has been updated
  - May need to change GET call to work around aggressive IE caching

23

Polling.js

```
function getData( ){
    if (xHRObject.readyState == 4)
          ...           // Load XML and transform XML
          ...  // Update HTML
           setTimeout("getDocument()", 5000);
  }
```

Clear the object and call the getDocument function in 5 seconds

```
function getDocument( ){
    xHRObject.open("GET", "GetStocksList.aspx?id=" + Number(new Date()), true);
    xHRObject.onreadystatechange = getData;
    xHRObject.send(null);
}
```

Unique value to get around IE caching of GET calls

24

Handling Server-side Errors

- Two strategies to getting non-response
  - Cancel Request
  - Retrying
- Approach
  - Test **`readystate`** and **`status`** separately
  - **`readystate = 4`** indicates response loaded
  - **`status = 200`** indicates HTTP status is OK
  - However, transient conditions on the internet can mean **`readystate = 4`** and **`status != 200`** , then

  => **`readystate`** will not be reset

25

Handling Errors – cancelling request

```
function getData( ){
    if ((xHRObject.readyState == 4)) {
        if (xHRObject.status == 200) {
            var serverText = xHRObject.responseText;
            ....
        }
        else {
            //testErrorCode
            span.innerHTML =    testErrorCode (xHRObject.status);
            xHRObject.abort();
        }
    }
}
```

Server return response OK

Clear Http object

Create function to give message indicating server side problem based on Http codes e.g.
201 created
400 bad request
401 unauthorised
403 forbidden
404 not found
500 internal server error
503 service unavailable

Handling Errors – retrying

```
function getData( ){
    if ((xHRObject.readyState == 4)) {
        if (xHRObject.status == 200) {
            var serverText = xHRObject.responseText;
            ....
        }
        else {
            span.innerHTML = "File X does not exist ";
            xHRObject.abort();
            if (retryCount < RETRY_MAX){
                setTimeout("sendRequest( )" , 5000);
                retryCount ++;
            }
            else
                //display error message}
        }}}
```

Callback function to sendRequest()

Server return response OK

Clear Http object

Retry a number of times

27

Using Regular Expressions

Web Developers should understand the concepts and value of using Regular Expressions

- Regular Expressions are a useful way to concisely define the syntax and 'pattern' of textural data.
- Simple functions can be used to test or 'match' data against the 'pattern'.
- Regular Expressions can be used in both client-side and server-side scripts, so the same 'pattern' can be consistently applied to verify data formats.

And in particular be able to:

- Use Regular Expressions to check values entered in HTML forms.

http://www.php.net/manual/en/book.pcre.php

28

What are Regular Expressions?

- are strings that describe the 'pattern' or 'rules' for strings
- are strings that follow a set of syntax rules
- can be used as a concise and consistent way to test for matching patterns
- *are great for checking form values!*

29

Regular Expressions in PHP

- PHP uses Perl Compatible Regular Expressions (PCRE) and has a range of pre-defined PCRE functions
  http://www.php.net/manual/en/ref.pcre.php
- Perform a regular expression match `preg_match()`
- Initialise a Regular Expression pattern, and test string

```
$pattern = "/(chapter \d+(\.\d)*)/i";
$str = "For more information, see Chapter 3.4.5.1";
if (preg_match($pattern, $str) {
    echo "A match was found.";
} else {
    echo "A match was not found.";
}
```

- Other functions `preg_replace(), preg_split()`

30

Regular Expressions in PHP

- A simple regular expression can be the equivalent of many lines of code.
- Simply define the 'pattern' of the Regular Expression, e.g. 'pattern' for a phone number such as

  (03) 9214-8000

  you could use

  ```
  $pattern = "/^\(\d\d\) \d\d\d\d-\d\d\d\d$/" ;
  ```

- Then 'match' the input string against the 'pattern'

  ```
  preg_match($pattern, $inputString)     // true if OK
  ```

31

Regular Expressions - Basic Examples

/WebProg/	matches "Isn't WebProg great?"
/^WebProg/	matches "WebProg rules!", not "What is WebProg?"
/WebProg$/	matches "I love WebProg", not "WebProg is great!"
/^WebProg$/	matches "WebProg", and nothing else
/bana?na/	matches "banana" and "banna", but not "banaana".
/bana+na/	matches "banana" and "banaana", but not "banna".
/bana*na/	matches "banna", "banana", and "banaaana", but not "bnana"
/^[a-zA-z]+$/	matches any string of one or more letters and nothing else.

32

Regular Expressions – More Examples

/[A-Za-z0-9-]+/	letters, numbers and hyphens
/\d{1,2}\/\d{1,2}\/\d{4}/	date such as 19/9/2006
/\S+\.(jpg|gif|png)	jpg, gif or png filename
/^[1-9]{1}$\|^[1-2]{1}[0-9]{1}$|^30$/	any number from 1 to 30
/#?([A-Fa-f0-9]){3}(([A-Fa-f0-9]){3})?/	valid hex colour code
/(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,15}/	password string (at least 1 uppercase letter, 1 lowercase letter and 1 digit)
/^.+@.+\..{2,3}$/	possible email address (in US)
/\<(/?[^\>]+)\>/	HTML tags

33

Regular Expressions - Basic Syntax

- Quantifiers
  - * 0 or more
  - + 1 or more
  - ? 0 or 1
  - {4} exactly 4
  - {4,} 4 or more
  - {4,6} 4,5 or 6
- Groups & Ranges
  - . Any character (except \n)
  - (a|b) a or b
  - (abc) group of abc (a followed by b followed by c)
  - [abc] set (“range”) a, b or c
  - [^abc] not a, b or c
  - [a-g] set range a to g
  - [3-6] set range of digits 3,4,5 and 6

34

Regular Expressions - Basic Syntax

- Pattern Basics

^	Start of string
$	End of string
.	Match any single character
(a|b)	a or b
(...)	Group section
[abc]	match any character in the set
[^abc]	not match in the set
[a-z]	match the range
\d	match a single digit from 0 to 9 *shortcut for [0-9]*
\D	not a digit

- Pattern Quantifiers

a?	0 or 1 of a
a*	0 or more of a
a+	one or more instance of a
a{3}	exactly 3 a's = aaa
a{3,}	3 or more a's
a{3,6}	between 3 to 6 a's
!(pattern)	"not" pattern

[\ ^ $. | ? * + ( ) are the 11 meta-characters, or special characters, used in the syntax.
If you want to include these, you need to escape them with \

eg. \(and \.

35

Regular Expressions - Basic Syntax

- Pattern Modifiers

/g	global matching
/i	case insensitive
/s	single line mode
/m	multiple-line mode
/x	allow comments and white space in pattern
/e	evaluate replacement
/U	ungreedy replacement

- Assertions

(?=	look ahead positive
(?!	look ahead negative
(?<=	look behind positive
(?<!	look behind negative

There are many useful online syntax references about Regular Expressions, such as:
http://www.php.net/manual/en/reference.pcre.pattern.syntax.php

36

Regular Expressions - Basic Syntax

- Anchors
  - ^ start of string
  - \A start of string
  - $ end of string
  - \Z end of string
  - \b word boundary
  - \B not word boundary
  - \< start of word
  - \> end of word
- Special Characters
  - \n newline
  - \r carriage return
  - \t tab
  - \v vertical tab
  - \f form feed
  - \xhh hex character hh
  - \s whitespace charater
  - \S not a whitespace character

Swinburne Library eBooks, can provide examples of Regular Expressions in many different scripting languages, e.g. search for 'regular expressions' in http://proquest.safaribooksonline.com/search

37

Object-Oriented Programming

- Object-oriented programming (OOP) refers to the creation of reusable software objects (or components) that can be easily incorporated into multiple programs
- An object refers to programming code and data that can be treated as an individual unit or component
- *Data* refers to information contained within variables or other types of storage structures
- The functions associated with an object are called *methods*
- The variables that are associated with an object are called *properties* or attributes
- Popular object-oriented programming languages include C++, Java, and Visual Basic

38

Encapsulation

- Objects are **encapsulated** – all code and required data are contained within the object itself
- Encapsulated objects hide all internal code and **data**
- An **interface** refers to the **methods** and **properties** that are required for a source program to communicate with an object
- Encapsulated objects allow users to see *only* the methods and properties of the object that you allow them to see
- Encapsulation reduces the complexity of the code
- Encapsulation prevents other programmers from accidentally introducing a bug into a program, or stealing code

39

Classes

- The code, methods, attributes, and other information that make up an object are organized into **classes**
- An **instance** is an object that has been created from an existing class
- Creating an object from an existing class is called **instantiating** the object
- An object **inherits** its methods and properties from a class — it takes on the characteristics of the class on which it is based
- **Inheritance** *also* allows you to build new classes based on existing classes without having to rewrite the code contained in the existing one

40

Using Classes in PHP Scripts

- Use a class to create (**instantiate**) an object in PHP through the new operator with a class constructor

```
$objectName = new ClassName();
```

- A **class constructor** is a special function with the same name as its class that is called automatically when an object from the class is *instantiated.* Parameters can be passed to the function

```
$checking = new BankAccount(01234587, 1021, 97.58);
```

- After an object is instantiated, use *member selection notation* - a hyphen and a 'greater than' symbol (->) to access the *methods* and *properties* contained in the object.

```
$b = $checking-> balance;
$cash = 100;
$checking-> withdrawal($cash);
```

41

Working with Database Connections as Objects

- Access MySQL database connections as objects by instantiating an object from the `mysqli` class
- To connect to a MySQL database server using *procedural syntax:*

```
$dbConnect = mysqli_connect("localhost", "dongosselin",
            "rosebud", "real_estate");
```

- To connect to a MySQL database server using *object-oriented syntax:*

```
$dbConnect = new mysqli("localhost", "dongosselin",
            "rosebud", "real_estate");
```

42

Selecting a Database and Closing a Database Connection

- *Procedural syntax* :

```
$dbConnect = mysqli_connect("localhost", "dongosselin", "rosebud");
mysqli_select_db($dbConnect, "real_estate");
// additional statements that access or manipulate the database
mysqli_close($dbConnect);
```

- *Object-oriented* version of the code :

```
$dbConnect = new mysqli("localhost", "dongosselin", "rosebud");
$dbConnect->select_db("real_estate");
// additional statements that access or manipulate the database
$dbConnect->close();
```

43

Handling MySQL Database Connection Error

- With object-oriented style, you cannot terminate script execution with the `die()` or `exit()` functions, in the one statement, as you would with *procedural syntax*

```
$dbConnect = @mysqli_connect("localhost", "dongosselin", "rosebud")
      or die("<p>Unable to connect to the database server.</p>"
      . "<p>Error code " . mysqli_connect_errno()
      . ": " . mysqli_connect_error()) . "</p>";
```

- With *object-oriented style*, check whether a value is assigned to the **`connect_errno`** or **`connect_error`** properties and *then* call the `die()` function to terminate script execution

```
$dbConnect = @new mysqli("localhost", "dgosselin", "rosebud");
if ($dbConnect->connect_error)
      die("<p>Unable to connect to the database server.</p>"
      . "<p>Error code " . $dbConnect->connect_errno
      . ": " . $dbConnect->connect_error . "</p>";
```

44

Handling Other MySQL Errors

- For any *methods* of the `mysqli` class that fail (as indicated by a return value of false), terminate script execution by appending `die()` or `exit()` functions to method call statements

```
$dbName = "guitars";
@$dbConnect->select_db($dbName)
     or die("<p>Unable to select the database.</p>"
     . "<p>Error code " . $dbConnect->errno . ": "
     . $dbConnect->error . "</p>");
```

45

Executing SQL Statements

- With object-oriented style, use the `query()` method of the `mysqli` class
- To return the fields in the current row of a resultset into an indexed array use:
  - The `fetch_row()` method of the mysqli class
- To return the fields in the current row of a resultset into an associative array use:
  - The `fetch_assoc()` method of the mysqli class

46

Executing SQL Statements (continued)

```
$sqlString = "SELECT * FROM inventory";
$queryResult = $dbConnect->query($sqlString)
     or die("<p>Unable to execute the query.</p>"
           . "<p>Error code " . $dbConnect->errno
           . ": " . $dbConnect->error . "</p>");
echo "<table width='100%' border='1'>";
echo "<tr><th>Make</th><th>Model</th>
      <th>Price</th><th>Inventory</th></tr>";
$row = $queryResult->fetch_row();
while ($row) {
     echo "<tr><td>{$row[0]}</td>";
     echo "<td>{$row[1]}</td>";
     echo "<td class='right'>{$row[2]}</td>";
     echo "<td class='right'>{$row[3]}</td></tr>";
     $row = $queryResult->fetch_row();
}
```

47

Defining PHP Classes

- The functions and variables defined in a class are called **class members**
- Class variables are referred to as **data members** or **member variables** *(properties in slide 38)*
- Class functions are referred to as **member functions** or **function members** *(methods in slide 38)*
- To create a class in PHP, use the `class` keyword to write a **class definition** that contains the data members and member functions that make up the class

```
class ClassName {
data member and member function definitions
}
```

48

Creating a Class Definition

- The ***ClassName*** portion of the class definition is the name of the new class. Class names usually begin with an uppercase letter to distinguish them from other identifiers.

```
class BankAccount {
  data member and member function definitions
}
```

- Use the **get_class** function to return the name of the class of an object

```
$checking = new BankAccount();
echo 'The $checking object is instantiated from the '
        . get_class($Checking) . " class.</p>";
```

- Use the **instanceof** operator to determine whether an object is instantiated from a given class

```
If($checking instanceof BankAccount)
  echo 'The $checking object is an instance of BankAccount’;
```

49

Collecting Garbage

- Garbage collection refers to cleaning up or reclaiming memory that is reserved by a program
- PHP knows when your program no longer needs a variable or object and automatically cleans up the memory for you
- The one exception is with open database connections

50

Using Access Specifiers

- Access specifiers control other programs' (or clients') access to individual data members and member functions
- There are three levels of access specifiers in PHP: `public`, `private`, and `protected`
- The **`protected`** access specifier allows access only *within* the class itself and by inherited and parent classes.
- The **`public`** access specifier allows anyone to call a class's member function or to modify a data member
- The **`private`** access specifier prevents clients from calling member functions or accessing data members and is one of the key elements in information hiding
- Private access does not restrict a class's internal access to its own members
- Private access restricts clients from accessing class members

51

Example: Using Access Specifiers

```
class BankAccount {
     public $balance = 958.20;
     public function withdrawal($amount) {
          $this->balance -= $amount;
     }
}
if (!class_exists("BankAccount")) {
     echo "<p>The BankAccount class is not available!</p>";
} else {
     $checking = new BankAccount();
     printf("<p>Your checking account balance is $%.2f.</p>",
     $checking->balance);
     $cash = 200;
     $checking->withdrawal($cash);
     printf("<p>After withdrawing $%.2f, your checking
          account balance is $%.2f.</p>",
          $cash, $checking->balance);
}
```

52

Initializing with Constructor Functions

- A **constructor function** is a special function that is called automatically when an object from a class is *instantiated*. It takes precedence over a function with the same name as the class.

```
class BankAccount {
      private $accountNumber;
      private $customerName;
      private $balance;
      function __construct() {
            $this->accountNumber = 0;
            $this->balance = 0;
            $this->customerName = "";
      }
}
```

- Constructor functions are commonly used in PHP to handle database connection tasks.

53

Cleaning Up with Destructor Functions

- A **default** constructor function is called when a class object is first instantiated
- A **destructor** function is called when the object is destroyed
- A destructor function cleans up any resources allocated to an object after the object is destroyed
- A destructor function is commonly called in two ways:
  - When a script ends
  - When you manually delete an object with the `unset()` function
- To add a destructor function to a PHP class, create a function named `__destruct()`

54

Example: Constructor / Destructor Functions

```
function __construct() {
   $dbconnect = new mysqli("localhost", "dongosselin",
             "rosebud", "real_estate")
}
function __destruct() {
   $dbConnect->close();
}
```

55

Writing Accessor Functions

- Accessor functions are public member functions that a client can call to retrieve or modify the value of a data member
- Accessor functions often begin with the words “set” or “get”
- Set functions modify data member values
- Get functions retrieve data member values

56

Writing Accessor Functions (continued)

```
class BankAccount {
     private $balance = 0;
     public function setBalance($newValue) {
          $this->balance = $newValue;
     }
     public function getBalance() {
          return $this->balance;
     }
}
if (!class_exists("BankAccount")) {
     echo "<p>The BankAccount class is not available!</p>";
} else {
     $checking = new BankAccount();
     $checking->setBalance(100);
     echo "<p>Your checking account balance is "
          . $checking->getBalance() . "</p>";
}
```

57

Serializing Objects

- Serialization refers to the process of converting an object into a string that you can store for reuse
- To serialize an object, pass an object name to the **`serialize()`** function

```
$savedAccount = serialize($checking);
```

- To convert serialized data back into an object, you use the **`unserialize()`** function

```
$Checking = unserialize($SavedAccount);
```

For more info, see “serialize”:
http://php.net/manual/en/function.serialize.php

58