COMP721 Web Development

AUT UNIVERSITY

Week 6: Managing State Information and OOP

Agenda

- *Understanding state information*
- *save state information*
  - *Using hidden form fields*
  - *Using query strings*
  - *Using HTML5 local storage*
  - *Using cookies*
  - *Using PHP sessions*
- *OOP in PHP*

6.1

UNDERSTANDING STATE INFORMATION

Understanding State Information

- Information about individual visits to a Web site is called **state information**
- HTTP was originally designed to be **stateless** – Web browsers store no persistent data about a visit to a Web site
- **Maintaining state** means to temporarily/persistently store information about a web user and its web site visits, that can be passed backwards and forwards between the client and the server.
- 'Temporary' means the info is lost when the browser is closed

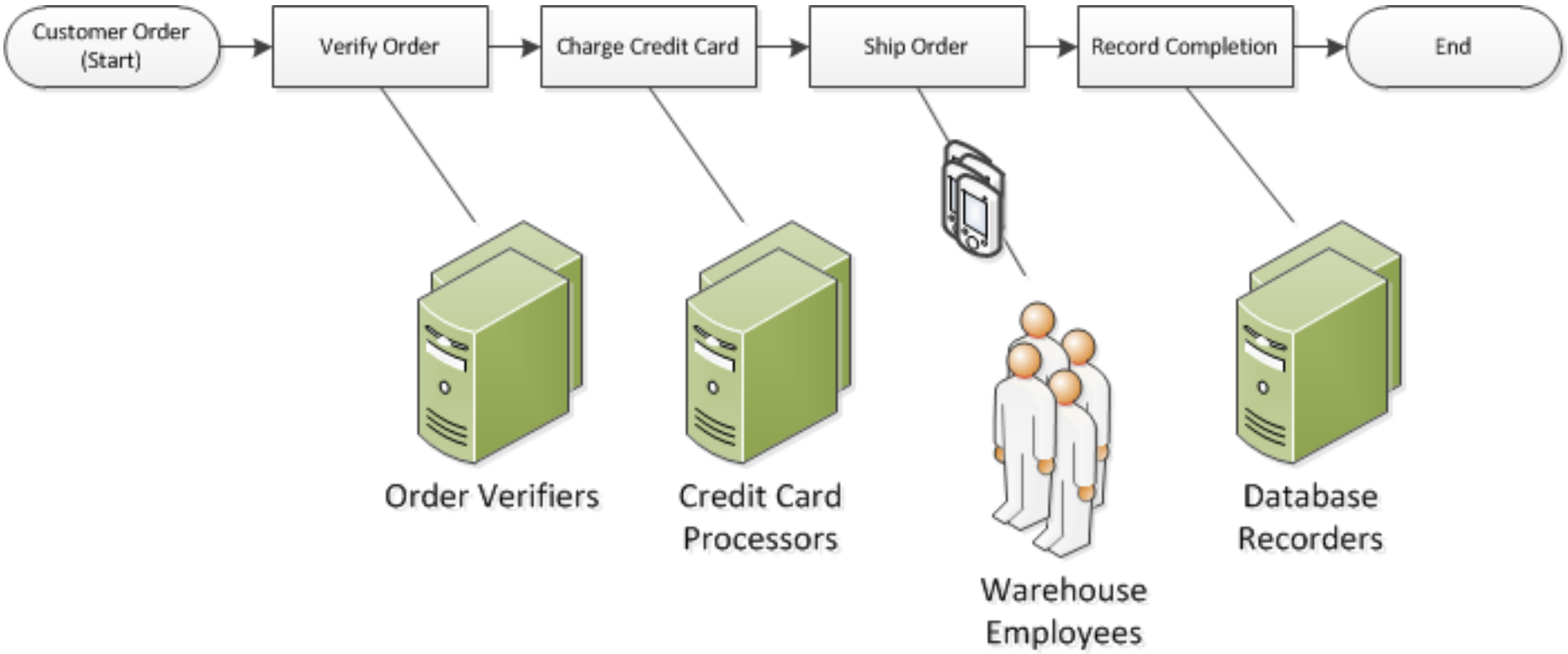
Why some web applications need state information?

Some more reasons why a web application may need to **maintain state** information:

- Temporarily store information for a user as a browser navigates within a multipart form (multiple webpages)
- Provide shopping carts that store order information
- Customize individual Web pages based on user preferences
- E-Commerce: order states

In general, state info is the info shared among a set of user-server interactions…

E-Commerce scenario



Discussion: For permanent state information, where can we store state information?

Client-side only? server-side only? Or on both sides?

Techniques for maintaining state information

- Hidden form fields
- Query strings
- Cookies
- HTML5 local storage (needs JavaScript)
- PHP Sessions – high-level constructs in PHP

SAVING STATING INFORMATION

Using Hidden Form Fields to Save State Information

- Hidden form fields temporarily store data that needs to be sent to a server that a user does not need to see
- Examples include the result of a calculation
- Create hidden form fields with the `<input />` element
- The syntax for creating hidden form fields is:

```
<input type="hidden" ... />
```

A relay game between client and server…

Using Hidden Form Fields to Save State Information (continued)

- Hidden form field attributes are **name** and **value**
- When submitting a form to a PHP script, access the values submitted from the form with the `$_GET[]` and `$_POST[]` superglobals
- To pass form values from one PHP script to another PHP script, store the values in hidden form fields

Using Hidden Form Fields

to Save State Information (continued)

```
<form action="courseListings.php" method="get">
<p>
<input type="submit" value="Register for Classes" />
<input type="hidden" name="diverID"
            value="<?php echo $DiverID ?>" />
</p>
</form>
```

Using Query Strings

to Save State Information

- A **query string** is a set of name=value pairs appended to a target URL
- A **query string** consists of a single text string containing one or more pieces of information
- Any forms that are submitted with the `GET` method automatically add a question mark (?) and append the **query string** to the URL of the server-side script

Using Query Strings

to Save State Information (continued)

- To pass information from one Web page to another using a query string,
  - add a question mark (?) immediately after the URL
  - followed by the query string containing the information in name=value pairs, and
  - separate the name=value pairs within the query string by ampersands (&)

```
<a href="page2.php?firstName=Don&lastName=Gosselin
&occupation=writer">Link Text</a>
```

Using Query Strings

to Save State Information (continued)

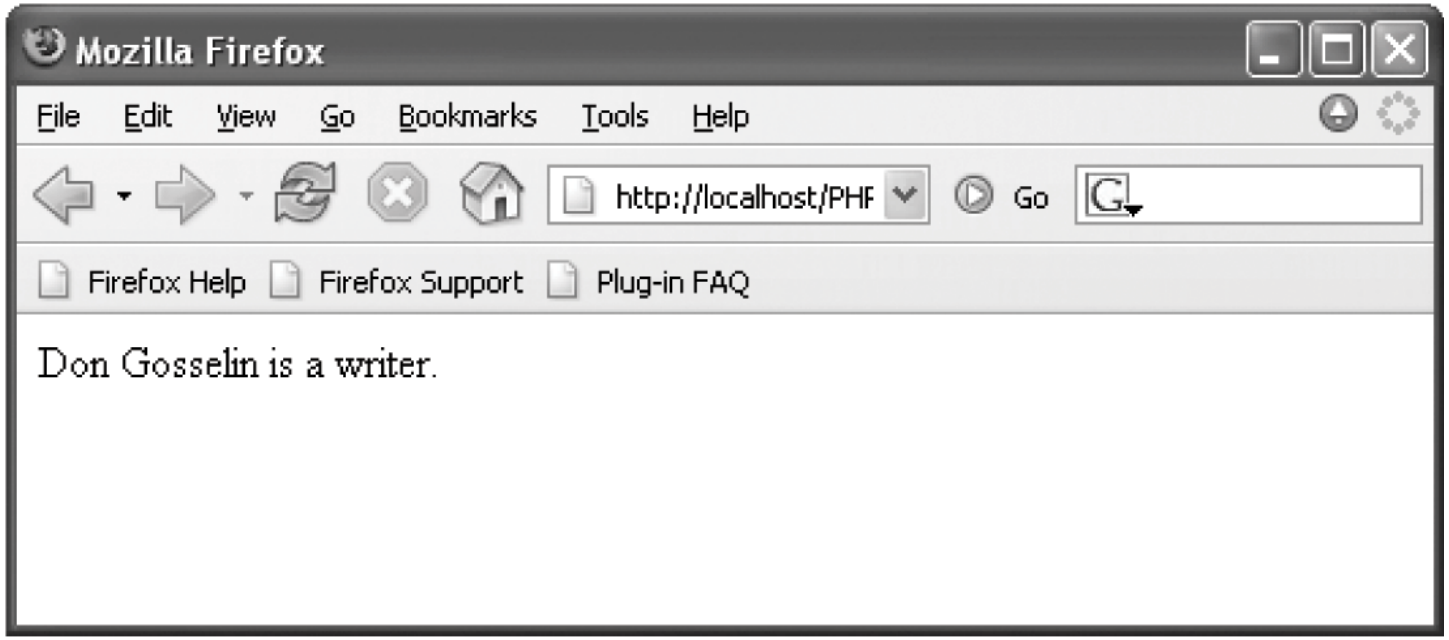
- To pass query string information from one PHP script to another PHP script, echo the values in the script

```
<a href="page2.php?firstName="<?php echo $fname ?>
"&lastName="<?php echo $lname ?>
"&occupation="<?php echo $occ ?>">Link Text</a>
```

Using Query Strings

to Save State Information (continued)

```
echo "{$_GET['firstName']} {$_GET['lastName']}
 is a {$_GET['occupation']}. ";
```



Output of the contents of a query string

- *Week 5 review*
- *save state information*
  - *Using hidden form fields*
  - *Using query strings*
  - *Using cookies*
  - *Using HTML5 local storage*
  - *Using PHP sessions*

6.2

- *OOP in PHP*

Using Cookies

to Save State Information

- Query strings do not permanently maintain state information
- After a Web page that reads a query string closes, the query string is lost
- To store state information beyond the current Web page session, Netscape created cookies
- **Cookies** are small pieces of information about a user that are stored by a Web server in text files on the user's computer

Using Cookies to Save State Information (continued)

- **Temporary cookies** remain available only for the current browser session
- **Persistent cookies** remain available beyond the current browser session and are stored in a text file on a client computer
- RFC 6265: A browser should be able to save "*at least 3000 cookies total; at least 50 cookies per unique host or domain name*" (http://www.faqs.org/rfcs/rfc6265.html)
- The largest cookie size is 4 kilobytes

Using Cookies: Creating Cookies

- The syntax for the `setcookie()` function is:

```
setcookie(name [,value ,expires, path, domain, secure])
```

- You must pass each of the arguments in the order specified in the syntax
- To skip the `value`, `path`, and `domain` arguments, specify an empty string as the argument value
- To skip the `expires` and `secure` arguments, specify 0 as the argument value

Using Cookies:Creating Cookies (continued)

- Call the `setcookie()` function before sending the Web browser any output, including white space, HTML elements (including the <html>), or output from the `echo()` or `print()` statements. As cookies are sent in the HTTP header
- Users can choose whether to accept cookies that a script attempts to write to their system
- A value of true is returned even if a user rejects the cookie

Using Cookies Creating Cookies (continued)

- Cookies cannot include semicolons or other special characters, such as commas or spaces, that are transmitted between Web browsers and Web servers using HTTP
- Cookies ***can*** include special characters when created with PHP since encoding converts special characters in a text string to their corresponding hexadecimal ASCII value

Using Cookies: `name` and `value` Arguments

- Cookies created with only the `name` and `value` arguments of the `setcookie()` function are *temporary cookies* because they are available for only the current browser session

No “expires” argument, for temporary cookies

```
<?php
setcookie("firstName", "Don");
?>
<htm>
 <head>
   <title>Skyward Aviation</title>
...
```

Using Cookies: `name` and `value` Arguments (continued)

- The `setcookie()` function can be called multiple times to create additional cookies –
- It's a good practice to put the `setcookie()` statements before any output on a Web page

```
setcookie("firstName", "Don");
setcookie("lastName", "Gosselin");
setcookie("occupation", "writer");
```

Using Cookies: `expires` Argument

- The `expires` argument determines how long a cookie can remain on a client system before it is deleted
- Cookies created without an `expires` argument are available for only the current browser session
- To specify a cookie's expiration time, use PHP's `time()` function

```
setcookie("firstName", "Don", time()+3600);
```

This "expires" argument, is set to current time + 3600 seconds

Using Cookies: `path` Argument

- The `path` argument determines the availability of a cookie to other Web pages on a server
- Using the `path` argument allows cookies to be shared within a server
- A cookie is available to all Web pages in a specified path as well as all subdirectories in the specified path
- Default value: the current directory that the cookie is being set in

```
setcookie("firstName", "Don", time()+3600,
   "/marketing/");

setcookie("firstName", "Don", time()+3600, "/");
```

Using Cookies: `domain` Argument

- The `domain` argument is used for sharing cookies across multiple servers in the same domain
- Cookies *cannot* be shared outside of a domain

```
setcookie("firstName", "Don", time()+3600,
  "/", ".gosselin.com");
```

Using Cookies: `secure` Argument

- The `secure` argument indicates that a cookie can only be transmitted across a secure Internet connection using HTTPS or another security protocol
- To use this argument, assign a value of 1 (for true) or 0 (for false) as the last argument of the `setcookie()` function

```
setcookie("firstName","Don",time()+3600,
  "/", ".gosselin.com", 1);
```

Using Cookies: Reading Cookies

- Cookies that are available to the current Web page are automatically assigned to the `$_COOKIE` autoglobal
- Access each cookie by using the cookie name as a key in the associative `$_COOKIE[]` array

```
echo $_COOKIE['firstName'];
```

- Newly created cookies are *not available* until after the current Web page is reloaded

Using Cookies: Reading Cookies

(continued)

- To ensure that a cookie is set before you attempt to use it, use the `isset()` function

```
setcookie("firstName", "Don");
setcookie("lastName", "Gosselin");
setcookie("occupation", "writer");
if (isset($_COOKIE['firstName'])
     && isset($_COOKIE['lastName'])
     && isset($_COOKIE['occupation']))
     echo "{$_COOKIE['firstName']}
          {$_COOKIE['lastName']}
          is a {$_COOKIE['occupation']}.";
```

Using Cookies: Reading Cookies

(continued)

- Can use multidimensional array syntax to set and read cookie values

```
setcookie("professional[0]", "Don");
setcookie("professional[1]", "Gosselin");
setcookie("professional[2]", "writer");
if (isset($_COOKIE['professional']))
     echo "{$_COOKIE['professional'][0]}
          {$_COOKIE['professional'][1]} is a
          {$_COOKIE['professional'][2]}.";
```

Using Cookies: Deleting Cookies

- To delete a persistent cookie before the time assigned to the `expires` argument elapses, assign a new expiration value that is sometime in the past
- Do this by subtracting any number of seconds from the `time()` function

```
setcookie("firstName", "", time()-3600);
setcookie("lastName", "", time()-3600);
setcookie("occupation", "", time()-3600);
```

Using HTML5 local storage

- Use JavaScript to get/set data in browser local storage, then use AJAX to asynchronously send data to PHP scripts.
- The following is an example in jquery

```
$.ajax({
    type: "POST",
    url: "example.php",
    data: { storageValue:
localStorage.getItem("yourValue"); }
});
```

Using PHP Sessions

to Save State Information

- A **session** refers to a period of activity when a PHP script stores *state information on a Web server*
- **PHP Sessions are** high-level language constructs that are implemented using cookies or query strings
- Configure your PHP engine (php.ini):
  - session.use_cookies: default “1”
  - session.use_only_cookies: : default “1”

http://ie.php.net/manual/en/session.configuration.php#ini.session.use-cookies

Starting a Session

- The `session_start()` function starts a new session or continues an existing one
- The `session_start()` function generates a unique session ID to identify the session
- A **session ID** is a random alphanumeric string that looks something like:

  `7f39d7dd020773f115d753c71290e11f`

- The `session_start()` function creates a text file on the Web server that is the same name as the session ID, preceded by `sess_`

Starting a Session (continued)

- Session ID text files are stored in the Web server directory specified by the `session.save_path` directive in your php.ini configuration file
- The `session_start()` function does not accept any functions, nor does it return a value that you can use in your script

```
<?php

session_start();

...
```

Starting a Session (continued)

- It is a good practice to call the `session_start()` function before you send the Web browser any output
- If a client's Web browser is configured to accept cookies, the session ID is assigned to a temporary cookie named `PHPSESSID`
- Pass the session ID as a query string or hidden form field to any Web pages that are called as part of the current session

Starting a Session (continued)

```
<?php
session_start();
...
?>
<p><a href='<?php echo "occupation.php?PHPSESSID="
      . session_id() ?>'>Occupation</a></p>
```

Working with Session Variables

- Session state information is stored in the `$_SESSION` autoglobal
- When the `session_start()` function is called, PHP either initializes a new `$_SESSION` autoglobal or retrieves any variables for the current session (based on the session ID) into the `$_SESSION` autoglobal

Working with Session Variables (continued)

```
<?php
session_set_cookie_params(3600);
session_start();
$_SESSION['firstName'] = "Don";
$_SESSION['lastName'] = "Gosselin";
$_SESSION['occupation'] = "writer";
?>
<p><a href='<?php echo "Occupation.php?"
 . session_id() ?>'>Occupation</a></p>
```

Sets the "lifetime" argument to 3600 seconds

Can we save an array in the session?

Working with Session Variables (continued)

- Use the `isset()` function to ensure that a session variable is set before you attempt to use it

```
<?php
session_start();
if (isset($_SESSION['firstName']) &&
   isset($_SESSION['lastName'])
      && isset($_SESSION['occupation']))
    echo "<p>" . $_SESSION['firstName'] . " "
         . $_SESSION['lastName'] . " is a "
         . $_SESSION['occupation'] . "</p>";
?>
```

Deleting a Session

- To delete a session manually, perform the following steps:
  1. Execute the `session_start()` function
  2. Use the `array()` construct to reinitialize the `$_SESSION` autoglobal
  3. Use the `session_destroy()` function to delete the session

Deleting a Session (continued)

```
<?php
session_start();
$_SESSION = array();//unset all session variables
session_destroy();
?>
```

4. Modify a “Registration” / “Log In” page so it deletes any existing user sessions whenever a user opens it.

6.3 OOP in PHP: Using Classes in PHP Scripts

Using Classes in PHP Scripts

- Use a class to create an object in PHP through the **new** operator with a class constructor
- 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*
- The syntax for *instantiating* an object is:

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

Using Classes in PHP Scripts (continued)

- The identifiers for an object name:
  - Must begin with a dollar sign ($)
  - Can include numbers or an underscore (but cannot start with a number)
  - Cannot include spaces
  - Are case sensitive

```
$checking = new BankAccount();
```

  - Can pass arguments to many constructor functions

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

Using Classes in PHP Scripts (continued)

- After an object is instantiated, use a hyphen and a 'greater than' symbol (**->**) to access the *methods* and *properties* contained in the object
- Together, these two characters **->** are referred to as ***member selection notation***
- With member selection notation append one or more characters to an object, followed by the name of a method or property

Using Classes in PHP Scripts (continued)

- With methods, include a set of parentheses at the end of the method name, just as with functions
- Like functions, methods can also accept arguments

```
$checking->getBalance();
$checkNumber = 1022;
$checking->getCheckAmount($checkNumber);
```

Mysqli also has an OO interface

Selecting a Database (continued)

- Example of *procedural syntax* to open a connection to a MySQL database server:

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

- An *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();
```

Defining Custom PHP Classes

Defining Custom PHP Classes

- **Data structure** refers to a system for organizing data
- The functions and variables defined in a class are called **class members**

Java terms

- Class variables are referred to as **data members** or **member variables** *(data fields)*
- Class functions are referred to as **member functions** or **function members** *(methods)*

Creating a Class Definition

- To create a class in PHP, use the `class` keyword to write a class definition
- A **class definition** contains the data members and member functions that make up the class
- The syntax for defining a class is:

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

Creating a Class Definition (continued)

- 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
- Within the class's curly braces, declare the data type and field names for each piece of information stored in the structure

```
class BankAccount {
data member and member function definitions
}
$Checking = new BankAccount();
```

Creating a Class Definition (continued)

- Class names in a class definition are ***not*** followed by parentheses, as are function names in a function definition
- 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'
```

Storing Classes in External Files

- PHP provides the following functions that allow you to use external files in your PHP scripts:
  - `include()`
  - `require()`
  - `include_once()`
  - `require_once()`
- You pass to each function the name and path of the external file you want to use

Using Access Specifiers/Modifiers

- Include an access specifier at the beginning of a data member declaration statement

```
class BankAccount {
        public $balance = 0;
}
```

- Always assign an initial value to a data member when you first declare it

```
class BankAccount {
        public $balance = 1 + 2;
}
```

Using Access Specifiers (continued)

- Create **public** member functions for any functions that clients need to access
- Create **private** member functions for any functions that clients do not need to access
- Also **protected**

Using Access Specifiers (continued)

```
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);
}
```

Initializing with Constructor Functions

- A **constructor function** is a special function that is called automatically when an object from a class is *instantiated*

```
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

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

Cleaning Up with Destructor Functions (continued)

- 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()`

Cleaning Up with Destructor Functions (continued)

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

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

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>";
}
```

Serializing Objects

- **Serialization** refers to the process of converting an object into a string that you can store for reuse
- Serialization stores both data members (properties) and member functions (methods) into strings
- To serialize an object, pass an object name to the **`serialize()`** function

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

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

Serializing Objects (continued)

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

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

- To use serialized objects between scripts, assign a serialized object to a session variable

```
session_start();
$_SESSION('SavedAccount') = serialize($Checking);
```

- Serialization is also used to store the data in large arrays.
  (Thus enabling a serialized array to be assigned, as a string, to a session variable.)

Serialization Functions

- When you serialize an object with the `serialize()` function, PHP looks in the object's class for a special "magical" function named `__sleep()`
- The primary reason for including a `__sleep()` function in a class is to specify which data members of the class to serialize

Serialization Functions (continued)

- If you do not include a `__sleep()` function in your class, the `serialize()` function serializes all of its data members

```
function __sleep() {
      $SerialVars = array('balance');
      return $SerialVars;
}
```

- When the `unserialize()` function executes, PHP looks in the object's class for a special "magical" function named `__wakeup()`

For more info, see "Magic Methods":
http://php.net/manual/en/language.oop5.magic.php