SWINBURNE UNIVERSITY OF TECHNOLOGY

Advanced Web Development: Object-Oriented Programming in PHP

Week 10

Outline

- Developing Object-Oriented PHP scripts
- Using Classes in PHP scripts
- Defining Custom PHP Classes
  - Initializing with Constructor Functions
  - Cleaning Up with Destructor Functions
  - Writing Accessor Functions
  - Serializing Objects with Serialization Functions
- Reading: Textbook Chapter 10
  PHP: Object-Oriented Programming, Classes and Objects
  http://php.net/manual/en/language.oop5.php

DEVELOPING OBJECT-ORIENTED PHP SCRIPTS

SWINBURNE UNIVERSITY OF TECHNOLOGY

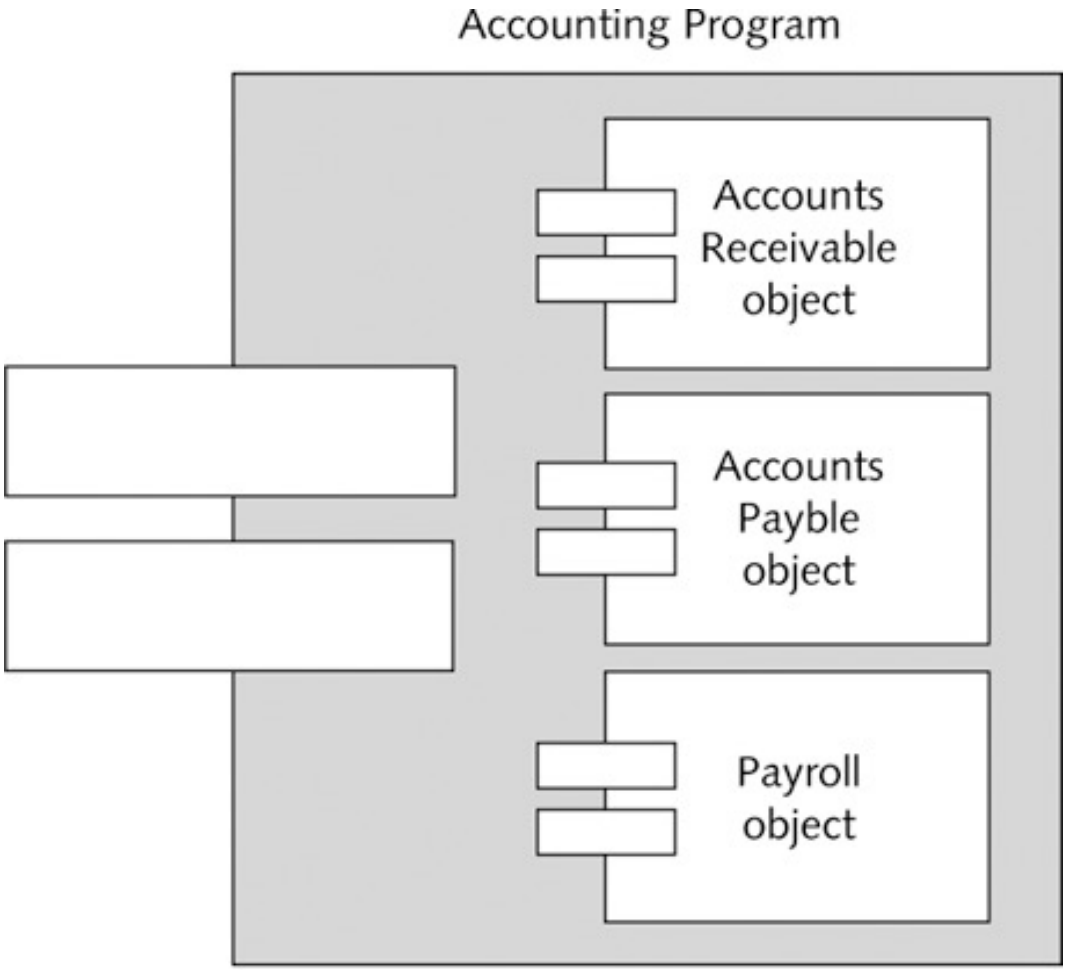
Object-Oriented Programming

- **Object-oriented programming** (OOP) refers to the creation of reusable software objects 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
- Objects are often also called **components**

Object-Oriented Programming (continued)

- ***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

Object-Oriented Programming (continued)



Accounting program

Object-Oriented Programming (continued)

Understanding 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

Object-Oriented Programming (continued)

- 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

Object-Oriented Programming (continued)

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

USING CLASSES IN PHP SCRIPTS

SWINBURNE UNIVERSITY OF TECHNOLOGY

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);
```

Using Classes in PHP Scripts (continued)

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 style:*

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

Using Classes in PHP Scripts (continued)

Instantiating and Closing a MySQL Database Object

- This statement also uses the `mysqli()` constructor function to *instantiate* a `mysqli` class object named `$dbConnect`

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

- To explicitly close the database connection, use the `close()` *method* of the `mysqli` class

```
$dbConnect->close();
```

Using Classes in PHP Scripts (continued)

Selecting a Database

- To connect to a MySQL database using *procedural syntax:*
  - Select or change a database with the `mysqli_select_db()` function
  - Pass two arguments to the `mysqli_select_db()` function:
    1. The variable representing the database connection
    2. The name of the database you want to use

Using Classes in PHP Scripts (continued)

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();
```

Using Classes in PHP Scripts (continued)

Handling MySQL Errors

- 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>";
```

Using Classes in PHP Scripts (continued)

Handling MySQL Errors (continued)

- 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>";
```

Using Classes in PHP Scripts (continued)

Handling MySQL Errors (continued)

- 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>");
```

Using Classes in PHP Scripts (continued)

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

Using Classes in PHP Scripts (continued)

Executing SQL Statements (continued)

```
$tableName = "inventory";
$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();
}
```

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

- *To avoid confusion – see also the terminology in slide 4*

Defining Custom PHP Classes (continued)

- Classes:
  - Help make complex programs easier to manage
  - Hide information that users of a class do not need to access or know about
  - Make it easier to reuse code or distribute your code to others for use in their programs
- *Inherited characteristics* allow you to build new classes based on existing classes without having to rewrite the code contained in the existing one

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

Storing Classes in External Files (continued)

- `include()` and `require()` functions both insert the contents of an external file, called an **include file**, into a PHP script
- `include_once()` and `require_once()` functions only include an external file *once* during the processing of a script
- Any PHP code must be contained within a PHP script section (`<?php ... ?>`) in an external file

Storing Classes in External Files (continued)

- Use the `include()` and `include_once()` functions for HTML code
- Use the `require()` or `require_once()` functions for PHP code
- External files can be used for classes and for any type of PHP code or HTML code that you want to reuse on multiple Web pages
- You can use any file extension you want for include files, *however servers are often configured to only parse PHP code in files with a .php extension*
- ***Work through the "PHP Server Side Includes Lab"***

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

Information Hiding

- **Information hiding** states that any class members that other programs, sometimes called clients, do not need to access or know about should be hidden
- Helps minimize the amount of information that needs to pass in and out of an object
- Reduces the complexity of the code that clients see
- Prevents other programmers from accidentally introducing a bug into a program by modifying a class's internal workings

Using Access Specifiers

- **Access specifiers** control a client's 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

Using Access Specifiers (continued)

- 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

Using Access Specifiers (continued)

- 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

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 = "";
        }
```

Initializing with Constructor Functions (continued)

- The `__construct()` function takes precedence over a function with the same name as the class
- 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

Summary

- Object-oriented programming (OOP) refers to the creation of reusable software objects 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
- Objects are often also called components
- Data refers to information contained within variables or other types of storage structures

Summary (continued)

- Objects are encapsulated – all code and required data are contained within the object itself
- The code, methods, attributes, and other information that make up an object are organized into classes
- Creating an object from an existing class is called instantiating the object

Summary (continued)

- 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
- Garbage collection refers to cleaning up or reclaiming memory that is reserved by a program
- Information hiding states that any class members that other programmers, sometimes called clients, do not need to access or know about should be hidden