Object-Oriented PHP

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

LEARNING TOPICS

- Introduction to OOP's
  - Understanding Encapsulation
  - OOP and Class
  - Using objects in PHP Scripts
  - Working with database connections as Objects
  - Handling MYSQL Errors
- Executing SQL Statements
- Defining Custom PHP Classes
  - Creating Class Definition
  - Storing Classes in External Files
  - Data Hiding
- Using Access Specifiers
- Serializing Objects
- Working With Member Functions
- Serialization Functions

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

- **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

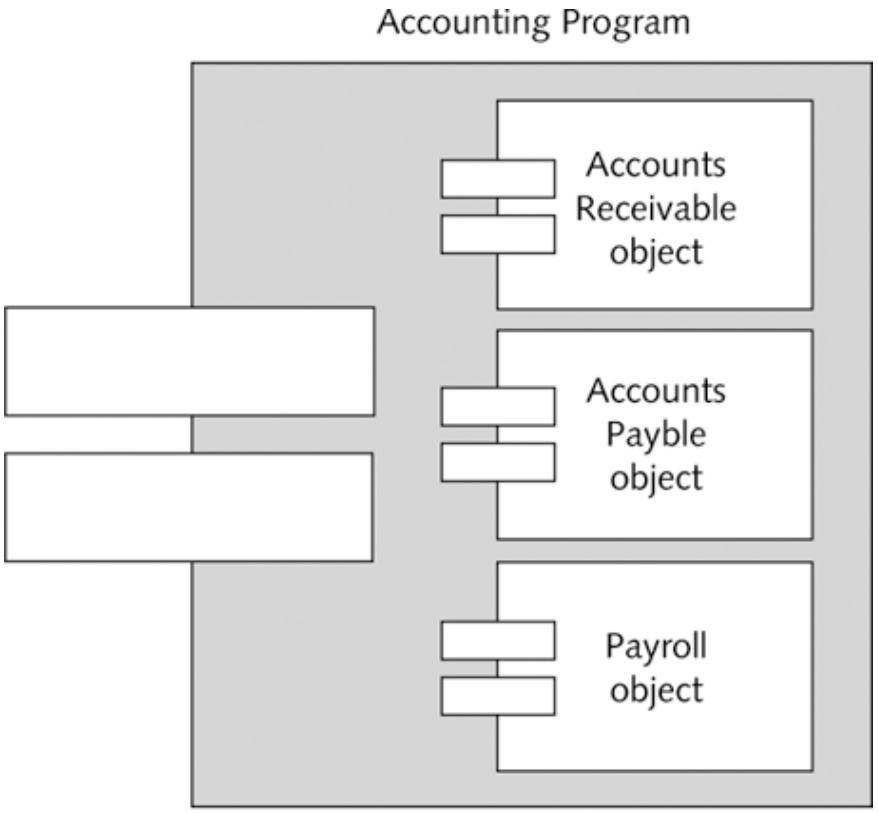

Figure 11-1 Accounting program

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

Understanding Encapsulation

- 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 and 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 Objects in PHP Scripts

- Declare an object in PHP by using 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 Objects in PHP Scripts

- The identifiers for an object name:
  - Must begin with a dollar sign
  - Can include numbers or an underscore
  - Cannot include spaces
  - Are case sensitive

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

  - Can pass arguments to many constructor functions

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

Using Objects 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 Objects 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);
```

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:

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

- To connect to the MySQL database server using object-oriented style:

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

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();
```

Selecting a Database

- 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

Selecting a Database (continued)

- An object-oriented version of the code:

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

Handling MySQL Errors

- With object-oriented style, you cannot terminate script execution with the `die()` or `exit()` functions

```
$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>";
```

Handling MySQL Errors

- With object-oriented style, check whether a value is assigned to the `mysqli_connect_errno()` or `mysqli_connect_error()` functions and then call the `die()` function to terminate script execution

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

Handling 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($DBName)
      Or die("<p>Unable to select the database.</p>"
      . "<p>Error code " . mysqli_errno($DBConnect)
      . ": " . mysqli_error($DBConnect)) . "</p>";
```

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 `mysqli_fetch_row()` function
- To return the fields in the current row of a resultset into an associative array use:
  - The `mysqli_fetch_assoc()` function

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 = $QueryResult->fetch_row();
do {
     echo "<tr><td>{$Row[0]}</td>";
     echo "<td>{$Row[1]}</td>";
     echo "<td align='right'>{$Row[2]}</td>";
     echo "<td align='right'>{$Row[3]}</td></tr>";
     $Row = $QueryResult->fetch_row();
} while ($Row);
```

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**
- Class functions are referred to as **member functions** or **function members**

Defining Custom PHP Classes

- 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

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

```
$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

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

- `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

- Use the `include()` and `include_once()` functions for files that will not prevent the application from running
- Use the `require()` or `require_once()` functions for files that will prevent the app from running if not present
- 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

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 programmers, 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 **`public` access specifier** allows anyone to call a class's member function or to modify a data member

Using Access Specifiers

- 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

- 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;
}
```

Serializing Objects

- **Serialization** refers to the process of converting an object into a string that you can store for reuse
  - This enables the sharing of objects within the same session used by multiple scripts
  - Session variables could be used but you would need to instantiate a new object and reassign the session variable values to the data members each time you call a script – this could be time consuming if the object has dozens of data members
- Serialization stores both data members and member functions into strings

Serializing Objects

- 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);
```

- Serialization is also used to store the data in large arrays
- To use serialized objects between scripts, assign a serialized object to a session variable

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

Working with Member Functions

- 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
- Access specifiers control a client's access to individual data members and member functions

Working with Member Functions

```
class BankAccount {
     public $Balance = 958.20;
     public function withdrawal($Amount) {
          $this->Balance -= $Amount;
     }
}
if (class_exists("BankAccount"))
     $Checking = new BankAccount();
else
     exit("<p>The BankAccount class is not available!</p>");
printf("<p>Your checking account balance is $%.2f.</p>",
     $Checking->Balance);
     $Cash = 200;
     $Checking->withdrawal(200);
     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

- 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

- 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()`

```
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"))
     $Checking = new BankAccount();
else
     exit("<p>The BankAccount class is not available!</p>");
$Checking->setBalance(100);
echo "<p>Your checking account balance is "
     . $Checking->getBalance() . "</p>";
```

Serialization Functions

- When you serialize an object with the `serialize()` function, PHP looks in the object's class for a special 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

- 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 function named `__wakeup()`

Serialization Functions

- Problem: Create a HitCounter class that counts the number of hits to a Web page and stores the results in a mySQL database'
- Use a private data member to store the number of hits and include public set and get member functions to access the private counter member variable

HitCounter.php

```
<?php
class HitCounter {
   private $DBConnect;
   private $DBName = "newdb";
   private $TableName = "hits";
   private $Hits = 0;
   function __construct() {
        $this->DBConnect = @new mysqli("localhost", "root", "mypassword");
        if (mysqli_connect_errno())
                die("<p>Unable to connect to the database server.</p>"
                . "<p>Error code " . mysqli_connect_errno()
                . ": " . mysqli_connect_error()) . "</p>";
   }
   function __destruct() {
        $this->DBConnect->close();
   }
   public function setDatabase($Database) {
        $this->DBName = $Database;
        @$this->DBConnect->select_db($this->DBName)
                Or die("<p>Unable to select the database.</p>"
                . "<p>Error code " . mysqli_errno($this->DBConnect)
                . ": " . mysqli_error($this->DBConnect)) . "</p>";
   }
```

HitCounter.php

```
    public function setTable($Table) {
          $this->TableName = $Table;
    }
    public function setHits() {
          $SQLstring = "UPDATE $this->TableName SET hits=$this->Hits WHERE
    id=1";
          $QueryResult = @mysqli_query($this->DBConnect, $SQLstring)
                  Or die("<p>Unable to perform the query.</p>"
                  . "<p>Error code " . mysqli_errno($this->DBConnect)
                  . ": " . mysqli_error($this->DBConnect)) . "</p>";
    }
    public function getHits() {
          $SQLstring = "SELECT * FROM $this->TableName WHERE id=1";
          $QueryResult = $this->DBConnect->query($SQLstring)
                  Or die("<p>Unable to perform the query.</p>"
                  . "<p>Error code " . mysqli_errno($this->DBConnect)
                  . ": " . mysqli_error($DBConnect)) . "</p>";
          $Row = $QueryResult->fetch_row();
          $this->Hits = $Row[1];
          $this->Hits = $this->Hits + 1;
          echo "<p>This page has received " . $this->Hits . " hits.</p>";
    }

}
```

CountVisits.php

```
<?php
require_once("HitCounter.php");
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>Hit Counter</title>
</head>
<body>
<h3>Hit Counter</h3>
<?php
$Database = "newdb";
$Table = "hits";
if (class_exists("HitCounter")) {
   $Counter = new HitCounter();
   $Counter->setDatabase($Database);
}
else
   exit("<p>The HitCounter class is not available!</p>");
$Counter->setTable($Table);
$Counter->getHits();
$Counter->setHits();
?>
</body>
</html>
```

Thank You

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283