

# ISIT307 - WEB SERVER PROGRAMMING

---

LECTURE 5.1 – PHP: OBJECT-ORIENTED PROGRAMMING – PART I



# LECTURE PLAN

---

- Study object-oriented programming concepts
- Use objects in PHP scripts
- Declare data members in classes
- Work with class member functions

# OBJECT-ORIENTED PROGRAMMING – KEY CONCEPTS

---

- object,
- encapsulation,
- association,
- aggregation,
- delegation,
- composition,
- dynamic binding,
- polymorphism,
- inheritance,
- hierarchical objects,
- abstract classes

# INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

---

- **Object-oriented programming (OOP)** refers the concept of merging related variables and functions into a single interface
- 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 orientedness can be considered as co-operative problem solving through objects communicating with one another.

# INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

---

- **Data** refers to information contained within variables or other types of storage structures
- The variables that are associated with an object are called **properties** or **attributes**
- The functions associated with an object are called **methods**

# 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 object **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

---

- In PHP the declaration of an object is by using the **new** operator

- The syntax for instantiating an object is:

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

- After an object is instantiated, a hyphen and a greater-than symbol (**->**) is used to access the methods and properties contained in the object
- Together, these two characters are referred to as **member selection notation**

# 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

# DEFINING CUSTOM 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** or **attribute** or **property**
- Class functions are referred to as **member functions** or **function members** or **operation** or **method**

# CREATING A CLASS DEFINITION

---

- To create a class in PHP, the `class` keyword is used in 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

---

- 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
- Example

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

# CREATING A CLASS DEFINITION

---

- Some useful functions:
  - `get_class()` function can be used to retrieve the name of the class
  - `class_exists()` can be used to determine if a class exists
  - `instanceof` operator can be used to determine whether an object is instantiated from a given class

```
$checking = new BankAccount();  
echo 'The $checking object is from' .  
    get_class($checking) . " class.</p>";  
---  
if ($checking instanceof BankAccount) {...}  
---  
if (class_exists("BankAccount")) {...}
```

# STORING CLASSES IN EXTERNAL FILES

---

- Good programming practice is to store the classes in separate file and include in the code using:
- `include()`, `require()`, `include_once()` or `require_once()` functions



# INFORMATION HIDING

---

- **Information hiding** states 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 and retrieve 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
- The **protected access specifier** prevents clients from calling member functions or accessing data members, but allows inherited (child) classes to have access

# USING ACCESS SPECIFIERS

---

- The access specifier must be included at the beginning of a data member declaration statement

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

- A data member should be assigned a value in the declaration

```
class BankAccount {  
    public $balance = 0;  
}  
---  
class BankAccount {  
    public $balance = 1 + 2; //invalid  
}
```

# WORKING WITH MEMBER FUNCTIONS

---

- **public** member functions should be declared all functions that clients need to access
- **private** member functions should be declared all functions that clients do not need to access
- If the member function is not assigned access specifiers, is by default public

# USING THE \$THIS REFERENCE

---

- Outside of a class, the members of the object can be referred by using the name of the object, the member selection notation (->), and the name of the function or variable
- Within a class function definition, \$this must be used to refer to the current object of the class

```
$this->accountNumber = 0;
```

# 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>");

echo "<p>Your checking account balance is \${$checking->balance}</p>";
$cash = 200;
$checking->withdrawal($cash);
echo "<p>After withdrawing \${$cash}, your checking account balance is
    \${$checking->balance}</p>";
```

# 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

---

- Constructor can have input arguments, or optional input arguments
- In the older version of PHP (PHP4) the constructor can be defined with the same name as the class, but it is deprecated in PHP7 and is not recognized as constructor in PHP8
- PHP 8.0 offers Constructor Property Promotion
  - allows class property declaration and constructor assignment right from the constructor



# INITIALIZING WITH CONSTRUCTOR FUNCTIONS

---

- Constructor Property Promotion

```
class BankAccount{  
    private float $balance;  
    public function __construct(float $balance) {  
        $this->balance = $balance;  
    }  
}
```

— — —

```
class BankAccount {  
    public function __construct(private float $balance) {}  
}
```

— — —

```
class BankAccount {  
    private string $name;  
    public function __construct(private float $balance, string $name) {  
        $this->name = $name; }  
}
```

# CLEANING UP WITH DESTRUCTOR FUNCTIONS

---

- 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
- The destructor syntax is:

```
function __destruct() { ... }
```

# WRITING ACCESSOR AND MUTATOR FUNCTIONS

---

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

# WRITING ACCESSOR AND MUTATOR FUNCTIONS - EXAMPLE

---

```
class BankAccount {
    private $balance=0;
    function __construct($bal=0)
    {
        $this->balance = $bal;
    }
    public function setBalance($newValue)
    {
        if ($newValue >0)
            $this->balance = $newValue;
    }
    public function getBalance()
    {
        return $this->balance;
    }
}

$checking = new BankAccount();
$checking->setBalance(100);
echo "<p>Your checking account balance is " . $checking->getBalance()
    . "</p>\n";

$newCh = new BankAccount();
$newCh->setBalance(-50);
echo "<p>Your checking account balance is " . $newCh->getBalance()
    . "</p>\n";
```

# WRITING ACCESSOR AND MUTATOR FUNCTIONS

---

- Also can be used the magic functions:
  - `__set()` – utilized when writing data to inaccessible (protected or private) data member/property
  - `__get()` – utilized for reading data from inaccessible (protected or private) data member/property

```
class MyClass{  
    private $myP;  
    function __get($name)  
    {        return $this->$name;}  
    function __set($name, $value)  
    {        $this->$name = $value; }  
}
```

```
$myV = new MyClass();  
$myV->myP = 5;  
echo $myV->myP;
```