

Apply Object Oriented Concepts in PHP

Unit - III

3.1 Creating Classes and Objects

- Class:

- ✓ A class is defined by using the *class* keyword, followed by the name of the class and a pair of curly braces.

Syntax:

```
class class_name  
{  
    //properties;  
    //methods( );  
}
```

3.1 Creating Classes and Objects

- Object:
 - ✓ Objects of a class is created using *new* keyword. Each object has properties and methods defined in the class.

Syntax:

```
$object_name = new class_name();
```

3.1 Creating Classes and Objects

- Object:

```
<html> <body>
<?php class student {
    public $name;
    function set_name($n) {
        $this->name = $n; }
    function get_name() {
        return $this->name;
    }
}
$stud1 = new student();
$stud1->set_name('Zaid');
echo $stud1->get_name();
?></body></html>
```

Output:
Zaid

3.1 Creating Classes and Objects

- Access Modifiers:

- ✓ There are 3 access modifiers in PHP.

- **Public:** Property and method can be accessed from everywhere. It is default.

- **Protected:** Property and method can be accessed within the class and child class.

- **Private:** Property and method can be accessed within the class only.

3.2 Constructor & Destructor

- Constructor:
 - ✓ Constructor allows you to initialize an object's properties upon creation of the object.
 - ✓ In PHP, a constructor function name is `__construct()`. It starts with an underscore (`_`).
 - ✓ PHP will automatically call constructor when an object for a class is created.

3.2 Constructor & Destructor

- Constructor:

```
<html> <body> <?php  
class student {  
    public $name;  
    function __construct($n) {  
        $this->name = $n; }  
  
    function get_name() {  
        return $this->name; }}  
$stud = new student("Zaid");  
echo $stud->get_name();  
?>  
</body></html>
```

Output:
Zaid

3.2 Constructor & Destructor

- Destructor:
 - ✓ Destructor is called when the object is destroyed or the script is stopped or exited.
 - ✓ In PHP, destructor function name is `__destruct()`. Function name start to two underscore (`__`).
 - ✓ PHP automatically call this function at the end of the script.

3.2 Constructor & Destructor

- Destructor:

```
<?php
class Fruit {
    public $name;
    public $color;
    function __construct($name) {
        $this->name = $name; }
    function __destruct() {
        echo "The fruit is {$this->name}."; }}
$apple = new Fruit("Apple");
?>
```

Output:

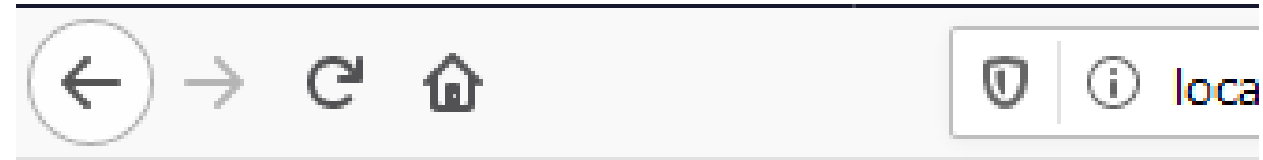
The fruit is Apple.

3.3 Inheritance

- When a class is derived from another class, it known as inheritance.
- The child class will inherit all the public and protected properties and methods of parent class.
- In addition, it can have its own properties and methods.
- An inherited class is defined by *extends* keyword.
- Final keyword can be used to prevent a class inheritance or to prevent method overriding.

3.3 Inheritance

```
<?php
class person
{
    public $name;
    public function __construct($n) {
        $this->name = $n; }
    public function intro() {
        echo "Hello "; } }
class Age extends person {
    public function message() {echo "<br> how are you today ".$this-
        >name."?"; } }
$a = new Age("zaid", "lecturer");
$a->intro();
$a->message(); ?>
```



Hello
how are you today zaid?

3.3 Inheritance

- *Abstract Class:*
 - An abstract class is a class that contains at least one abstract method.
 - An abstract method is a method that is declared but not implemented in the class.
 - Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.
 - An abstract class or method is defined with the *abstract* keyword:

3.3 Inheritance

```
<?php
abstract class student {    // Parent class
    public $name;
    public function __construct($name) {
        $this->name = $name; }
    abstract public function intro() : string; }

class computer extends student { // Child classes
    public function intro() : string {
        return "Hello.. This is $this->name, i am from computer department"; }}

$stud = new computer("Zaid");
echo $stud->intro();
?>
```

Output:
Hello.. This is Zaid, i am from computer department

3.3 Inheritance

- Static Methods:
 - Static methods can be called directly - without creating an instance of a class.
 - Static methods are declared with the keyword *static*.
 - To access a static method use *class_name*, double semi colon(*::*) and *method name*.

class_name :: static_method_name();

- A class can have both static and non static methods.
- A static method can be accessed from a method in the same class using the keyword *self* and double semi colon (*::*)

self::static_method_name();

- To call a static method from a child class, use the *parent* keyword inside child class. *parent::static_method_name();*

3.3 Inheritance

- Static Methods:

```
<?php
class greeting {
    public static function welcome() {
        echo "Hello World!";
    }
}
```

```
// Call static method
greeting::welcome();
?>
```

Output:
Hello World!

3.3 Inheritance

- Static Methods:

```
<?php
class greeting {
    public static function welcome() {
        echo "Hello World!";
    }
    public function __construct() {
        self::welcome();
    }
}
new greeting();
?>
```

Output:
Hello World!

3.3 Inheritance

- Static Methods:

```
<?php
class domain {
    protected static function getWebsiteName() {
        return www.MHSSP.in; }
}
class domainW3 extends domain {
    public $websiteName;
    public function __construct() {
        $this->websiteName = parent::getWebsiteName();}
}
$domainW3 = new domainW3;
echo $domainW3->websiteName;
?>
```

Output:

www.MHSSP.in

3.3 Inheritance

- Static Properties:
 - Static properties can be called directly - without creating an instance of a class.
 - Static properties are declared with the keyword *static*.
 - To access a static property use *class_name*, double semi colon(*::*) and *property name*.

class_name :: static_property_name;

- A class can have both static and non static properties.
- A static property can be accessed from a method in the same class using the keyword *self* and double semi colon (*::*)

self::static_property_name;

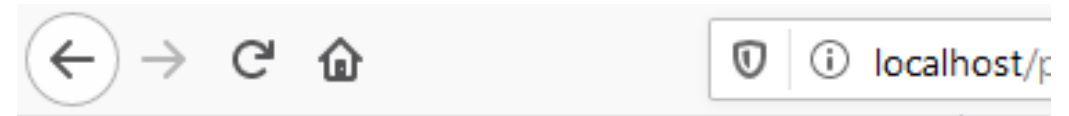
- To call a static property from a child class, use the *parent* keyword inside child class. *parent::static_property_name);*

3.3 Method Overloading

- In PHP, we can perform method overloading with the help of magic function `_call()`.
- The `_call()` function accepts two arguments: `function_name` & `argument (array)`.
- `_call()` is automatically invoked whenever a function call is given with `function_name` mentioned above.

3.3 Method Overloading

```
class shape {  
    function __call($name_of_function, $arguments) {  
        if($name_of_function == 'area') {  
            switch (count($arguments)) {  
                case 1: return 3.14 * $arguments[0];  
                case 2: return $arguments[0]*$arguments[1];  
            }  
        }  
    }  
}  
  
$s = new Shape;  
echo("Area of Circle: ".$s->area(2));  
echo "<br>";  
echo ("Area of rectangle: ".$s->area(4, 2));
```



Area of Circle: 6.28
Area of rectangle: 8

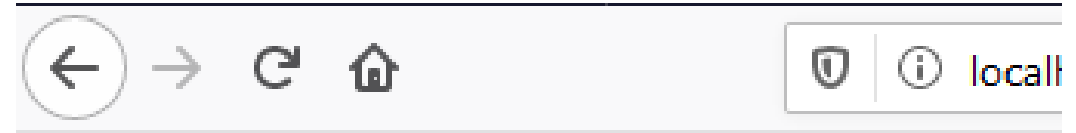
3.3 Method Overriding

- When a function is implemented in child class with same name & signature as that of its parent class, then the function in parent class is said to be overridden by the child class.
- This process is known as function/method overriding.

3.3 Method Overriding

```
<?php
class A
{ public function show()
  {   echo "<br>Inside Parent class"; } }
class B extends A
{ public function show()
  {   echo "<br>Inside Child class"; } }

$a=new A();
$b=new B();
$a->show();
$b->show();
?>
```



Inside Parent class
Inside Child class

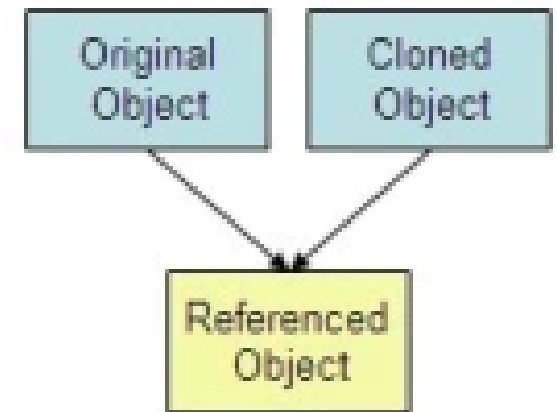
3.3 Final Keyword

- The *final* keyword is used only for methods and classes.
- If we define a method with final, then it prevent us from overriding the method.
- If final keyword is used with class, then it prevents the class from being inherited.

3.3 Object Cloning

- Shallow Copy:
 - In the process of shallow copying A, B will copy all of A's field values.
 - If the field value is a memory address it copies the memory address, and if the value is a primitive type it copies the value of the primitive type.
 - But in shallow copy, if you modify the content of B's field you are also modifying what A's field contains.

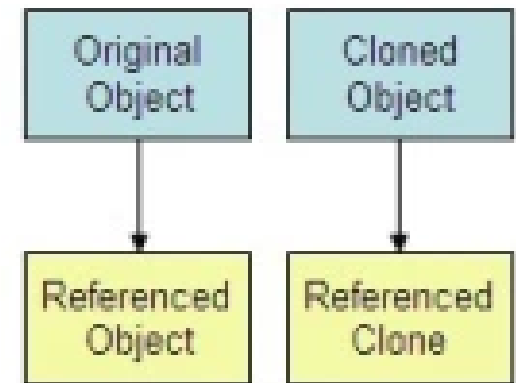
Shallow Clone



3.3 Object Cloning

- Deep Copy:
 - In this process, the data is actually copied completely.
 - The advantage is that A and B do not depend on each other.
 - In this method, all the things in object A's memory location get copied to object B's memory location.

Deep Clone



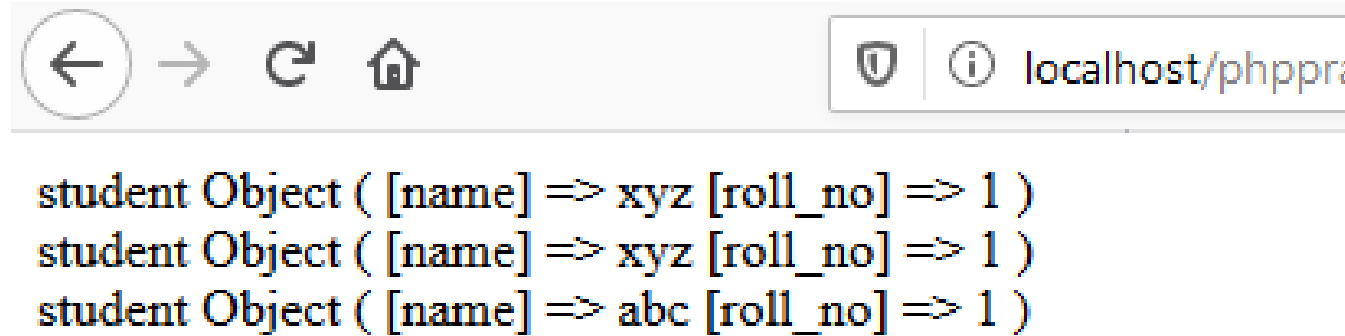
3.3 Object Cloning

- In PHP, when an object is copied using *assignment operator* (=) then a *shallow copy* is performed on the original object.
- To perform a *Deep copy*, *clone* keyword is used.

3.3

Object Cloning

```
<?php
class student
{ public $name;
  public $roll_no;
  function __construct($a,$b)
  { $this->name=$a;
    $this->roll_no=$b; } }
$stud1=new student("abc",1);
$stud2=$stud1;
$stud3=clone $stud1;
$stud1->name="xyz";
print_r($stud1); echo "<br>";
print_r($stud2); echo "<br>";
print_r($stud3);
?>
```



```
student Object ( [name] => xyz [roll_no] => 1 )
student Object ( [name] => xyz [roll_no] => 1 )
student Object ( [name] => abc [roll_no] => 1 )
```

3.4 Introspection

- Introspection, in PHP, is the ability to examine an object's characteristics, such as its name, parent class properties (if any), classes, interfaces and methods.
- PHP offers large number of functions that one can use to accomplish the task.

3.4 Introspection

Function Name	Description
<code>class_exists()</code>	Checks whether a class has been defined.
<code>get_class()</code>	Returns the class name of an object
<code>get_parent_class()</code>	Returns the class name of an object's parent class.
<code>is_subclass_of()</code>	Checks whether an object has a given parent class.
<code>get_declared_classes()</code>	Returns a list of all declared classes.
<code>get_class_methods()</code>	Returns the names of the class methods.
<code>get_class_vars()</code>	Returns the default properties of a class.
<code>interface_exists()</code>	Checks whether the interface is defined.
<code>method_exists()</code>	Checks whether an object defined a method.

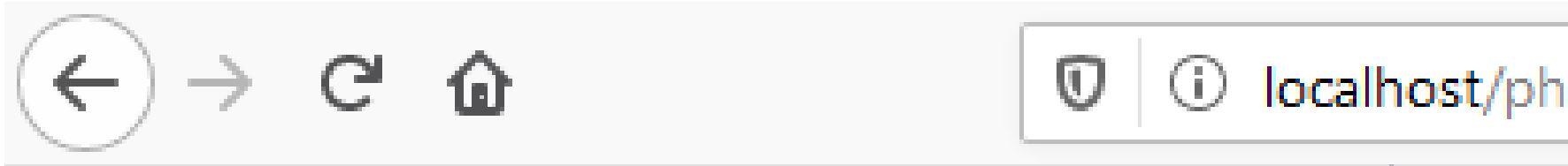
3.4 Introspection

```
<?php
class Introspection
{   public function description() {
        echo "I am a super class for the
        Child class.<br>";
    }}
class Child extends Introspection
{
    public function description() {
        echo "I'm " . get_class($this) , "
        class.<br>";
        echo "I'm " .
        get_parent_class($this) , "'s child.<br>";
    }}
if (class_exists("Introspection")) {
    $introspection = new Introspection();
    echo "The class name is: " .
    get_class($introspection) . "<br>";
    $introspection->description();}
if (class_exists("Child")) {
    $child = new Child();
    $child->description();
    if (is_subclass_of($child,
    "Introspection")) {
        echo "Yes, " . get_class($child) . " is
        a subclass of Introspection.<br>";
    }
    else {
        echo "No, " . get_class($child) . " is
        not a subclass of Introspection.<br>";
    }
}
```

Prepared By: Khan Mohammed Zaid, Lecturer, Comp. Engg., MHSS

30

3.4 Introspection



The class name is: Introspection
I am a super class for the Child class.
I'm Child class.
I'm Introspection's child.
Yes, Child is a subclass of Introspection.

3.4 Serialization

- It is a technique used by programmers to preserve their working data in a format that can later be restored to its previous form.
- Serializing an object means converting it to a bytestream representation that can be stored in a file.
- This is useful for persistent data; for example, PHP sessions automatically save and restore objects.

3.4 Serialization

- `Serialize()`:
 - It converts a storable representation of a value.
 - It accepts a single parameter which is the data we want to serialize and returns a serialized string.
 - A serialize data means a sequence of bits so that it can be stored in a file, a memory buffer or transmitted across a network link.

Syntax:
`serialize(value);`

3.4 Serialization

- `Unserialize()`:
 - It is used to create the original variable values.
 - i.e. converts actual data from serialized data.

Syntax:

`unserialize(serialized_string);`

3.4 Serialization

```
<?php
$s=serialize(array('Welcome','to','PHP'));
print_r($s."<br>");
$us=unserialize($s);
print_r($us);
?>
```

