Unit 3

Object Oriented Programming

1. Introduction

PHP provides extensive support for developing object-oriented web applications. Starting with PHP5, the object model was rewritten to allow for better performance and more features. This was a major change from PHP4. PHP5 has a full object model. Among the features in PHP5 are the inclusions of visibility, abstract and final classes and methods, additional magic methods, interfaces, cloning and typehinting.

Object Oriented Programming (OOP) is a programming concept that treats functions and data as objects.

Advantages of OOP

i. OOP provides a clear modular structure for programs.

ii. It is good for defining abstract data types.

iii. Implementation details are hidden from other modules and have a clearly defined interface.

iv. It is easy to maintain and modify existing code as new objects can be created with small differences to existing ones.

v. Objects, methods, instance, message passing, inheritance are some important properties provided by these particular languages.

vi. Encapsulation, polymorphism, abstraction are also counts in these fundamentals of programming language.

vii. In OOP, programmer not only defines data types but also deals with operations applied for data structures.

Features of Object Oriented Programming

i. More reliable software development is possible.

ii. Enhanced form of 'C' programming language.

iii. The most important feature is that it's procedural and object oriented nature.

iv. Much suitable for large projects.

2. Creating a Class

A Class is a unit of code composed of variables and functions which describes the characteristics and behavior of all the members of a set.

A class defines what an object will look like when it is created. It defines, *for example*, what the **methods** will do and what the **member** variables will be.

A class definition begins with the keyword *class*, followed by a class name, followed by a pair of curly braces which enclose the definitions of the class's properties and methods.

The class name can be any valid label which is a not a PHP *reserved word*.

A valid class name starts with a letter or underscore, followed by any number of letters, numbers, or under scores. A class may contain its own *constants*, *variables* (called 'properties'), and functions (called 'methods').

Syntax

```
class classname[extends baseclass]
{
   [var $property[ = value]; . . .]
   [function functionname(arguments)
   {// code
   }
   . . .
   ]
}
```

Example

```
<? php
class myclass
{
.................;
.................. ;
}
?>
```

This defines a class named myclass.

3. Creating an Object

To create an instance of a class, a new object must be created and assigned to a variable which is achieved with the keyword new. Consider the following example in which we are creating an object with the name as instance1 and instance 2 respectively for the class myclass which we have created.

```
<?php
$instance1 = new myclass();
// This can also be done with a variable:
$newname = 'hello';
$instance2 = new $newname();
?>
```

3.1 Adding a Method

The class myclass isn't useful if it isn't able to do anything, for this a method is required which is basically a function.

```
<?php
class myclass
{
 function mymethod()
 {
    print "hello you have called a method of class myclass";
 }
}
?>
```

An object derived from this class is now capable of printing out a message "hello you have called a method of class myclass" whenever the method mymethod is invoked.

To invoke the method on the object $instance1, we need to invoke the operator -> to access the newly created function mymethod.

```
<?php
$instance1=new myclass();
$instance->mymethod();
?>
```

The object is now capable of printing the greeting message "hello you have called a method of class myclass".

3.2 Adding a Property

Adding a property to a class is as simple as adding a method. We have to declare a variable inside the class to hold the value of the property. The variables are declared at the top of the class declaration.

```
<?php
class myclass
{
public $name;
function mymethod()
{
  print "hello $this->name";
}
}
?>
```

1

Apr. 2017 – 1M
Explain the purpose of $ this variable.

$this

The pseudo-variable *$this* is available when a method is called from within an object context. *$this* is a reference to the calling object (usually the object to which the method belongs, but possibly another object, if the method is called statically from the context of a secondary object).

Note

When accessing properties we need only one $. The syntax is $obj->property not $obj->$property. The property variable is declared as public and accessed using $ob->property.

Property declarations are optional. Consider the following *example* that has an undeclared $name property.

```
<?php
class myclass
{
   function myname()
   {
      return $this->name;
   }
   function setmyname($next_name)
   {
      $this->name=$next_name;
   }
}
?>
```

Default values can be assigned to properties, but those should only be constants.

```
var $name="john";
```

3.3 Visibility (Public, Private, Protected)

The visibility of class members, (properties, methods), relates to how that member may be manipulated within, or from outside the class. Three levels of visibility exist for class members.

i. Public

ii. Private

iii. Protected

By default, all class members are public.

With PHP3 or 4's object model, all class members were defined with the var keyword, which is equivalent to public in PHP5.

```
<?php
class MyClass
{
   public $publicMember = "Public member";
   protected $protectedMember = "Protected member";
   private $privateMember = "Private member";
   function myMethod()
```

```
    {
      // ...
    }
}
$obj = new MyClass();
?>
```

i. **Public:** Public class members (properties and methods) are available through-out the script and may be accessed from outside the class by using:

```
$obj->publicMember
```

and by accessing it from inside the myMethod method via the special $this variable. *For example,*

```
$this->publicMember
```

If another class inherits a public member, it can be accessed both from outside the derived class's objects and from within its methods.

Example

```
<?php
class test
{
   public $abc;
   public $xyz;
   public function xyz()
   {
   }
}
$objA = new test();
echo $objA->abc;//accessible from outside
$objA->xyz();//public method of the class test
?>
```

In the above *example*, class test is the very basic class. In this class everything is open. Minimum restriction in the class is to access its property and methods using object outside the class.

ii. **Private:** Private method or properties can only be accessible within the class. We cannot access private variable or function of the class by making object out of the class. But we can use private function and property within the class using $this object.

Private visibility in PHP classes is used when we do not want our property or function to be exposed outside the class.

Example

```
<?php
class Test
{
   public $abc;
   private $xyz;
   public function pubDo($a)
   {
      echo $a;
   }
      private function privDo($b)
      {
         echo $b;
      }
      public function pubPrivDo()
      {
         $this->xyz = 1;
         $this->privDo(1);
         }
      }
         $objt = new test();
         $objt->abc = 3;//Works fine
         $objt->xyz = 1;//Throws fatal error of visibility
         $objt->pubDo("test");//Prints "test"
         $objt->privDo(1);//Fatal error of visibility
         $objt->pubPrivDo();//Within this method private
         function privDo and //variable xyz is called
         using $this variable.
      ?>
```

iii. **Protected:** Protected visibilities in PHP classes are only useful in case of inheritance and interface. Protected method or variable can be accessible either within class or child class.

Example

```
<?php
class Parent
{
   protected $pr;
   public $a;
   protected function testParent()
   {
      "echo this is test";
   }
}
```

```
class Child extends Parent
{
   public function testChild()
   {
      $this->testParent(); //will work
   }
}
$objParent = new Parent();
$objParent->testParent();//Throws error
$objChild = new Child();
$objChild->setChild();//works because test child will call
test parent.
?>
```

In the above *example*, we can observe that method testParent() is not accessible from object of class. But it is accessible in child class.

4. Introspection

3

Apr. 2018 – *4M*
Write a short note on Introspection.
Oct. 17, Apr. 17 – *4M*
What is introspection? Explain any two introspective function.

Introspection is the ability of a program to examine an object's characteristics, such as its name, parent class (if any), properties, and methods.

With introspection, we can write code that operates on any class or object. We need not know which methods or properties are defined when we write the code; instead, it can discover that information at runtime, which makes it possible for us to write generic debuggers, serializers, profilers, etc.

4.1 Examining Classes

2

Oct. 17, Apr. 17 – *1M*
Name any two functions to extract basic information about classes in php?

To determine whether a class exists, use the **class_exists()** function, which takes in a string and returns a Boolean value.

Alternately, the get_declared_classes() function, returns an array of defined classes and checks if the class name is in the returned array:

```
$class = class_exists(classname);
$classes = get_declared_classes();
```

To get the methods and properties that exist in a class (including those that are inherited from superclasses) using the **get_class_methods()** and **get_class_vars()** functions. These functions take a class name and return an array:

```
$methods = get_class_methods(classname);
$properties = get_class_vars(classname);
```

The class name can be a bare word, a quoted string, or a variable containing the class name:

```
$class = 'student';
$methods = get_class_methods($class);
$methods = get_class_methods(student); // same
$methods = get_class_methods('student'); // same
```

The array returned by get_class_methods( ) is a simple list of method names. The associative array returned by get_class_vars( ) maps property names to values and also includes inherited properties.

Note get_class_vars( ) returns only properties that have default values; there's no way to discover uninitiailized properties.

Use get_parent_class() to find a class's parent class:

```
$superclass = get_parent_class(class name);
```

4.2 Examining an Object

To get the class to which an object belongs, first make sure it is an object using the is_object() function, then get the class with the get_class() function:

```
$obj = is_object(var);
$classname = get_class(object);
```

Before calling a method on an object, we can ensure that it exists using the method_exists() function:

```
$method_exists = method_exists(object,method);
```

get_object_vars(): returns an array of properties set in an object:

```
$array = get_object_vars(object);
```

get_object_vars(): returns only those properties that are set:

```
class student
{
   var $name;
   var $age;
}
$stud = new student;
$stud->name = 'hello';
$st = get_object_vars($stud); // $st is array('name' => 'hello');
```

The get_parent_class() function actually accepts either an object or a class name. It returns the name of the parent class, or FALSE if there is no parent class:

```
class A {}
class B extends A {}
$obj = new B;
echo get_parent_class($obj); // prints A
echo get_parent_class(B); // prints A
```

Consider the following *example* which exercises the introspection functions for classes and objects:

```
<?php
class stud
{
   var $name = 'aaa';
   var $add = 'xyz';
   var $age = 17;
   function first_fun() { }
   function second_fun() { }
}
class stud1 extends stud
{
  var $sex = false;
  function third_fun() { }
}
class stud2 extends stud1
{ }
$s = new stud;
$s->name = 'bbb';
$s->age = 23;
$s1 = new stud1;
$s1->name = 'ccc';
$s1->sex = true;
$s2 = new stud2;
print get_parent_class($s);
print get_parent_class($s1);
?>
```

5. Serialization

4

Apr.18, Oct.17, 16 – *1M*
What is serialization?

Apr. 2019 – *4M*
Write short notes on Serialization.

Serializing an object means converting it to a byte stream representation that can be stored in a file.

This is useful for persistent data; *for example*, PHP sessions automatically save and restore objects.

Serialization is most commonly used with PHP's sessions, which handle the serialization .This can be achieved by serialize and unserialize functions

serialize(): Returns a string containing a byte-stream representation of *value* that can be stored anywhere. This is useful for storing or passing PHP values around without losing their type and structure.

Syntax

```
$encode=serialize(something);
```

unserialize(): Takes a single serialized variable (see serialize( )) and converts it back into a PHP value. The converted value is returned, and can be a Boolean, integer, float, string, array or object. In case the passed string is not un-serializeable, `FALSE` is returned and `E_NOTICE` is issued.

Syntax

```
$something = unserialize(encoded);
```

For example, your pages might start like this:

```
<?php
include('object_definitions.inc'); // load object definitions
session_start(); // load persistent variables
?>
<html>...
```

PHP has two hooks for objects during the serialization and unserialization process: `__sleep()` and `__wakeup()`. These methods are used to notify objects that they're being serialized or unserialized.

The `__sleep()` method is called on an object just before serialization; it can perform any cleanup necessary to preserve the object's state, such as closing database connections, writing out

unsaved persistent data, and so on. It should return an array containing the names of the data members that need be written into the bytestream.

Conversely, the `__wakeup()` method is called on an object immediately after an object is created from a bytestream. The method can take any action it requires, such as reopening database connections and other initialization tasks.

Example, below is an object class, Log, which provides two useful methods: `write()` to append a message to the logfile, and `read()` to fetch the current contents of the logfile. It uses `__wakeup()` to reopen the logfile and `__sleep()` to close the logfile.

Example, the Log.inc file

```
<?php
class log
{
   var $fname;
   var $fp;
   function log($fname)
   { $this->fname = $fname;
   $this->open();
}
function open()
{
   $this->fp = fopen($this->fname, "a")
   or die("Can't open {$this->fname}");
}
function write($note)
{
   fwrite($this->fp, "$note\n");
}
function read()
{ return join('',file($this->fname));
}
function _ _wakeup()
{
   $this->open();
}
function _ _sleep()
{
   // write information to the account file
   fclose($this->fp);
```

```
    return array('filename');
}
}
?>
```

Store the Log class definition in a file called *Log.inc*. The HTML page in example, below uses the Log class and PHP sessions to create a persistent log variable, $l.

Example, front.php

```
<?php
include_once('log.inc');
session_start();
?>
<html><head><title>Front Page</title></head>
<body>
<?php
$now = strftime("%c");
if (!session_is_registered('l'))
{
   $l = new log("/tmp/persistent_log");
   session_register('l');
   $l->write("Created $now");
   echo("Created session and persistent log object.<p>");
}
$l->write("Viewed first page $now");
echo "The log contains:<p>";
echo nl2br($l->read());
//Insert line breaks where newlines (\n) occur in the string
?>
<a href="next.php">Move to the next page</a>
</body></html>
```

The output when this page is viewed is shown in

Output
The log contains
Created Fri April 4.08. 19:30 2010
Viewed First Page April 4.09. 19:30 2010
Move to the next pag

6. Inheritance (Extending a class)

Probably the greatest feature of the PHP OOP model is Inheritance.

Inheritance is the ability of PHP to extend classes (child classes) that inherit the characteristics of the parent class. The resulting object of an extend class has all the characteristics of the parent class, plus whatever is in the new child, or extended class.

A class can inherit the methods and properties of another class by using the keyword *extends in* the class declaration.

Note It is not possible to extend multiple classes; a class can only inherit from one base class.

The inherited methods and properties can be overridden by re-declaring them with the same name defined in the parent class. However, if the parent class has defined a method as final, that method may not be overridden.

It is possible to access the overridden methods or static properties by referencing them with **parent::**

Consider the following example,

```
<?php
class myclass
{
   // property declaration
   public $var = 'a default value';
   // method declaration
   public function displayVar()
{ echo $this->var;
}
}
class ExtendClass extends myclass
{
   // Redefine the parent method
   function displayVar()
   { echo "Extending class\n";
      parent::displayVar();
   }
}
$extend = new ExtendClass();
$extend->displayVar();
?>
```

Output
Extending class
a default value

6.1 Overriding

Overriding is a process of modifying the inherited method. So, in case of inheritance we only need to create method with same name in our child class which we want to override. Following is the *example* of overriding of method in PHP.

```
<?php
class testParent
{
  public function f1()
  {
    echo 1;
  }
  public function f2()
  {
    echo 2;
  }
  }
  class testChild
  {
    function f2($a)  //overriding function f2
    {
      echo "$a";
    }
    }
    $a = new testChild();
    $a->f2("vision "); //it will print vision
?>
```

In the above *example*, we are overriding function f2. While overriding we are free to change business logic, visibility and number of parameters.

Final

Any method or class that is declared as Final cannot be overridden or inherited by another class.

For example,

```
<?php
final class maths
{

} /*** end of class ***/

class modulo extends maths
{
}
?>
```

By running the above code we will get an error such as Fatal error: Class modulo may not inherit from final class (maths).This can protect us from those who wish to use our code for a purpose other than that for which it was intended.

7. Constructors and Destructors

3

Apr. 2019 – *1M*
Which is special function provided by PHP to define constructor?

Oct. 17, 16 – *5M*
Write a short note on Constructor.

Constructor is a function defined in our PHP class. Constructor function is automatically called when we create object of the class. As soon as we write $object = new myClass() our constructor function of the class will be executed. In PHP4 you can create constructor by creating function with same name of our class. But from PHP5 you can also create constructor by defining magic function _construct.

Syntax

```
void _construct([mixed $args[, $... ]])
```

Destructor is a function which is called right after you release an object. Releasing object means that you do not need it or use it anymore. This makes destructor suitable for any final actions you want to perform.

```
void _destruct(void)
```

Difference between Object Oriented Code in PHP4 and PHP5

PHP4	PHP5
Constructor is a function which has the same name as a class. *Example*, If you have a class named myclass, constructor is a function named myclass. PHP4 does not have destructors at all.	Constructor is a function called _ _construct. PHP5 is a backward compatible, so if it cannot find a function named _ _construct in a class declaration, it will search for a PHP4 style constructor. Destructor is a PHP5 feature.

<?php
class Student
{
var $name;
var $address;
var $phone;
// this is constructor
function student()
{
	<?php
	class student
	{
	var $name;
	var $address;
	var $phone;
	// this is constructor
	function _construct()
	{

```
      $this->name = "vision";
      $this->address = "xyz";
      $this->phone = 9989;
   }
     function printstudInfo()
{
      echo $this->name. "\n";
      echo $this->address. "\n";
      echo $this->phone. "\n";
     }
}
$stud = new student;
$stud->printstudInfo();
$stud = NULL;
?>
```

```
   $this->name = "vision";
   $this->address = "xyz";
    $this->phone = 9989;
}
    // this is destructor
    function _destruct()
{
   echo "student Object Released\n";
}
function printstudInfo()
{
   echo $this->name. "\n";
   echo $this->address. "\n";
   echo $this->phone. "\n";
 }
}
$stud = new student;
$stud->printstudInfo();
$stud = NULL;
?>
```

Passing Parameter in the Constructor

Constructors can take arguments and these arguments can be optional, which makes them much more useful. To be able to still use the class without parameters, all parameters to constructors should be made optional by providing default values.

```
<?php
class interestCalculator
{
   public $rate;
   public $duration;
   public $capital;
   //Constructor of the class
   public function _construct($rate, $duration)
   {
      $this->rate = $rate;
      $this->duration = $duration;
   }
   }
   $objCls = new interestCalculator(3.2,7)//passing value of $rate and
   // $duration
?>
```

If we have passed parameter to the constructor we need to pass value for them on the time of object creation.

```
$objCls = new interestCalculator(3.2,7).
```

If you will not send the parameters to the constructor, PHP will throw error.

7.1 References

References in PHP are a means to access the same variable content by different names.

Whenever we assign an object to another variable, a copy of it is created.

Example,

```
<?php
$first=5;
$ref=& $first;
for($i=0; $i<5; $i++)
{
   echo "The value of first is: $first, and the value of reference is:
   $ref<BR>";
   $first++;
   $ref--;
}
?>
```

We have used the = & operator to set the variable $ref as a reference to the variable $first.

Output

The value of first is: 5, and the value of reference is: 5
The value of first is: 5, and the value of reference is: 5
....
The value of first is: 5, and the value of reference is: 5
As we can see, neither variable seemed to move from its original value.

Step 1: Assigning the integer value 5 to the variable $first.

Step 2: We create a reference to the variable $first and call it $ref.

Step 3: We print the contents of both variables in the for loop and as expected, both are equal to five.

Step 4: Increment the value of $first, decrement the value of $ref, and repeat.

But why did the values of $first and $ref remain constant?

In the *example* we created a reference to $first and called it $ref. When this is done, a second variable is not created, but rather a single variable is given another variable name, as shown below:

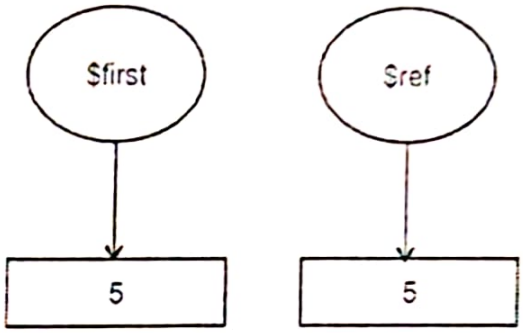

Figure 3.1: $first and $ref as independent variables

Consider another *example*,

```
<?php
$colors = 'black';
$settings['colors'] = &$colors; // Makes a reference
$colors = 'white'; // $colors changes
echo $settings['colors']; // Displays "white"
?>
```

Passing by reference allows us to keep the new variable "linked" to the original source variable. Changes to either the new variable or the old variable will be reflected in the value of both.

Consider the following *example* which stores data-related to changing the page's look and feel i.e. font size and color:

```
<?php
// set the $color and $size as per your choice
class Appearance
{
  var $color;
  var $size;
   function Appearance()
   {
      $this->color = 'Red';
      $this->size = 'small';
   }
function color_get()
{
   return $this->color;
}
function size_get()
   {
     return $this->size;
```

```
  }
  function color_set($color)
  {
    $this->color = $color;
  }
  function size_set($size)
  {
    $this->size = $size;
  }
}
// Output deals with building content for display
class FormOutput
{
  var $Appearance;
  var $output;

 // Constructor takes Appearance as its argument
  function FormOutput($Appearance)
  {
    $this->Appearance = $Appearance;
  }
  function buildOutput()
  {
     $this->output = 'Color is'. $this->Appearance->color_get().'and
     size is'.
     $this->Appearance->size_get();
  }
  function display()
  {
    $this->buildOutput();
    return $this->output;
  }
}
// Create an instance of Appearance
$Appearance = new Appearance();
// Pass it to an instance of Output
$output = new FormOutput($Appearance);

// Display the output
echo $output->display();
?>
```

This displays the following message

Color is Red and size is small

```
$Appearance = new Appearance();        // Create a Appearance
$output = new FormOutput($Appearance); // Pass it to an Output
// Modify some settings $Appearance->color_set('black');
$Appearance->size_set('medium');
// Display the output
echo $output->display();
 function FormOutput(&$Appearance)
 {
   $this->Appearance = &$Appearance;
 }
```

Notice that we have to use the reference operation twice here. This is because the variable is being passed twice-first to the constructor function, then again, to place it in a member variable.

Once we've made these changes, the display looks like this:

Color is black and size is medium

References inside the Constructor

Creating references within the constructor can lead to confusing results.

```
<?php
class refcon
{
   function refcon($name)
{ // create a reference inside the global array $globalref
     global $globalref;
     $globalref[] = &$this;
     // set name to passed value
     $this->setName($name);
     // and put it out
    $this->echoName();
   }
   function echoName()
{
  echo "<br />", $this->name;
  }
  function setName($name)
{
    $this->name = $name;
   }
```

```
}
$refconst = new refcon('set in constructor');
$refconst->echoName();
$globalref[0]->echoName();
?>
```

```
/*Output
set in constructor
set in constructor set in constructor */
$bar2 =& new refcon('set in constructor');
$bar2->echoName( );
$globalref[1]->echoName( );
/*Output
set in constructor
set in constructor
set in constructor */
?>
```

Apparently there is no difference, but in fact there is a very significant one:

$bar1 and $globalref[0] are _NOT_ referenced, they are NOT the same variable.

This is because 'new' does not return a reference by default, instead it returns a copy.

8. Interfaces

Interface provides a way for defining a way to which a class adheres; the interface provides method prototypes and constants, and any class that implements the interface must provide implementations for all methods in the interface.

Here the *syntax* for an interface definition:

```
interface interfacename
{
[ function functionname();
.....
]
}
```

To declare that a class implements an interface, include implements keyword and any number of interfaces, separated by commas:

```
interface Printable
{
   function poutput();
}
class icomponent implements Printable
{
   function poutput()
   {
      echo "printing an image …..";
}
}
```

An interface may inherit from other interfaces as long as none of the interfaces it inherits from declare methods with the same name as those declared in the child interface.

Note When a class uses/implements an interface, the class MUST define all the methods/functions of the interface otherwise the PHP engine will give you an error !!!!

Consider the following example,

```
<?php
class bike implements testdrive
{
public $color;
public $num_tyres;
public $price;
public $shape;
public $brand;
public function _construct()
{
   echo 'Bike details.<br/>';
}
public function showPrice()
{
   echo 'The Bike costs '.$this->price.'.<br/>';
}
public function numtyres()
{
   echo 'This Bike has '.$this->num_tyres.' tyres.<br/>';
}
public function drive()
{
  echo 'Bike Zooms catch me if u can!!!';
}
```

```
public function stop()
{
   echo 'STOP!!!<br/>';
}
}
interface testdrive
{
  function drive();
  function stop();
}
$object = new Bike;
$object->drive();
$object->stop();
?>
```

Primary Purposes of an Interface

i. Interfaces allow us to define/create a common structure for our classes – to set a standard for objects.

ii. An interface solves the problem of single inheritance – they allow us to inject 'qualities' from multiple sources.

iii. Interfaces provide *a flexible* base/root structure that we don't get with classes.

iv. Interfaces are great when we have multiple coders working on a project – we can set up a loose structure for programmers to follow and let them worry about the details.

9. Encapsulation

Encapsulation is about grouping of functionality (operations) and related data (attributes) together into a coherent data structure (classes). It is the ability to hide details of implementation.

Encapsulation refers to these two basic concepts

i. The protection of class internal data from code outside that class.

ii. The hiding of the details of implementation.

Consider the following example,

```
<?php
class arithmetic
{
  var $first=1000;
  var $second=500;
  function add()
  {
    $res=$this->first+$this->second;
    echo "Addition = ".$res."<br/>";
  }
  function sub()
  {
    $res=$this->first-$this->second;
    echo "Subtraction = ".$res."<br/>";
  }
  function mult()
  {
    $res=$this->first*$this->second;
    echo "Multiplication = ".$res."<br/>";
  }
  function div()
  {
    $res=$this->first/$this->second;
    echo "Division = ".$res."<br/>";
  }
}
$obj = new arithmetic();
$obj->add();
$obj->sub();
$obj->mult();
$obj->div();
?>
```

Output
Addition = 1500
Subtraction = 500
Multiplication = 500000
Division = 2

In the above example

Two properties $first and $second along with the methods add(), sub(), mult() and div() are defined in a single unit, i.e., in the class arithmetic. We create the object of class arithmetic. That object calls the methods and methods are performed their task with defined properties.

add() method performs addition of two properties $first and $second.

sub() method performs subtraction of two properties $first and $second.

mult() method performs multiplication of two properties $first and $second.

div() method performs division of two properties $first and $second.

Solved Programs

1. How would you declare a class called emptyclass() that has no methods or properties.

Solution

An empty class can be declared in the following way

```
<?php
class emptyclass {}
?>
```

2. Write a PHP program to check properties of class and print.

Solution

```
<?php
class Test
{   public    $p1 = 1;
    protected $p2 = 2;
    private   $p3 = 3;
}
$t1 = new Test();
$reflect = new ReflectionClass($t1);
$props=$reflect->getProperties(ReflectionProperty::IS_PUBLIC|ReflectionP
roperty::IS_PROTECTED);
foreach($props as $prop)
{
   print $prop->getName() . "\n";
}
var_dump($props);
?>
```

3. Write a PHP script to accept student details (number, name, class) and print it in sorted order of name.

Solution

HTML code is as follows

```
<form method ='POST' action ='student.php'>
rollno <input type='text' name='txtrollno'>
name<input type='text' name='txtname'>
class<input type='text' name='txtclass'>
Select the choice:
<select name='op'>
<option value=1>Save details of students</option>
<option value=2>Display the list of students in sorted order as per
name</option></select><br>
<input type='submit'><input type ='reset'>
</form>
```

PHP code is as follows

```
<?php
class student
{ private $rollno,$name,$class;
function_construct($rollno,$name,$class)
{ $this->rollno = $rollno;
$this->name = $name;
$this->class = $class;
}
function put_student()
{ printf("<tr><td>%d</td><td>%s</td><td>%s</td></tr>",$this-
>rollno,$this->name,$this->class);
}
function get_rollno()
{ return $this->rollno;
}
function get_name()
{ return $this->name;
}
function get_class()
{ return $this->class;
}
}
$rollno = $_POST['txtrollno'];
$name = $_POST['txtname'];
$class = $_POST['txtclass'];
```

```
$op = $_POST['op'];
switch($op)
{ case 1:
$fp = fopen("student1.dat","a");
$s = new student($code,$name,$class);
$encode = serialize($s);
fwrite($fp,$encode);
fwrite($fp,"\n");
fclose($fp);
echo "Record saved successfully.";
break;
case 2:
$fp = fopen("student1.dat","r");
$records = array();
while($encode = fgets($fp,200))
{
   $s = unserialize($encode);
   $records[] = $s;
}
fclose($fp);
$n = count($records);
for($i=0;$i<$n-1;$i++)
{
   for($j=0;$j<$n-$i-1;$j++)
   {
      $cmp=strcmp($records[$j]->get_name(),$records[$j+1]->get_name());
      if($cmp<0)
      { $t = $records[$j];
        $records[$j] = $records[$j+1];
        $records[$j+1] = $t;
      }
   }
}
echo "<table border=1>";
echo "<tr><th>Rollno</th><th>Name</th><th>Class</th></tr>";
for($i=0; $i<$n; $i++)
$records[$i]->put_student();
echo "</table>";
}
?>
```

4. Write a PHP program to display information of class and its objects.

Solution

```
<?php
class First
{
   var $v1;
   var $v2="hello";
   var $v3=1000;
   function First()
   {
      return(true);
   }
   function myfunction1()
   {
      return(true);
   }
   function myfunction2()
   {
      return(true);
   }
}
$f1 = new First();
$classname=get_class($f1);
echo "class name : $classname<br>";
echo "Class methods<br>";
$classmethods=get_class_methods('First');
foreach($classmethods as $methodname)
{
   echo "$methodname<br>";
}
echo "Class properties<br>";
$classproperties=get_class_vars($classname);
foreach($classproperties as $k => $v)
{
   echo "$k : $v<br>";
}
?>
```

5. Write a PHP script to create class shape and its subclasses triangle, square and circle and display the area of selected shape (use concept of inheritance).

Solution

class.html

```
<html>
<head>
<title>class inheritance example</title>
</head>
<body>
<form method = 'POST' action = 'class.php'>
Shape
<input type = 'radio' name = 'shape' value = 'Rectangle'> Rectangle
<br>
<input type = 'radio' name = 'shape' value = 'Square'> Square
<br>
<input type = 'radio' name = 'shape' value = 'Circle'> Circle
<br> <br>
Enter dimension1 : <input type = 'text' name = 'dim1'>
<br>
Enter dimension2 : <input type = 'text' name = 'dim2'>
<br> <br>
<input type = 'submit' value = 'Submit'> <input type = 'reset'
value = 'Reset'>
</form>
</body>
</html>
```

class.php

```
<?php
class Shape
{
  var $dim1,$dim2;
  function Shape($a,$b)
  {
    $this -> dim1 = $a;
    $this -> dim2 = $b;
  }
  function cal_area()
  {
    return($this -> dim1 * $this -> dim2);
  }
}
```

```
class Rectangle extends Shape
{
  function Rectangle($x,$y)
  {
    parent :: Shape($x,$y);
  }
  function cal_area()
  {
    return parent :: cal_area();
  }
}
class Square extends Shape
{
  function Square($x)
  {
    parent :: Shape($x,$x);
  }
function cal_area()
{
  return parent :: cal_area();
}
}
class Circle extends Shape
{
  function Circle($x)
  {
    parent :: Shape($x,$x);
  }
function cal_area()
{
  return( 3.14 * parent :: cal_area());
}
}
$d1 = $_POST['dim1'];
$d2 = $_POST['dim2'];
$sh = $_POST['shape'];
if($sh == "Rectangle")
{
  $obj = new Rectangle($d1,$d2);
}
else
if($sh == "Square")
```

```
{
  $obj = new Square($dl);
}
else
if($sh == "Circle")
{
  $obj = new Circle($dl);
}
$area = $obj -> cal_area();
print "The area of the selected shape $sh is $area sq.units <br>";
print "<a href = 'class.html'> Go Back </a>";
?>
```

6. Write a PHP script which will give details about all declared classes, their properties, methods within a program using nested for each loop.

Solution

```
<?php
//define base class
class Emp
{
  //define the properties
  public $name;
  public $age;
  //define some methods
  public function _construct()
  {
  echo "This is a Constructor for Employee class.\n";
  }
public function Empdetails()
{
echo "The Employee details method.\n";
}
}
// use reflection to inspect the class
$reflector= new ReflectionClass('Emp');
//list constants
echo "constants";
foreach($reflector->getConstants() as $key => $value)
{
echo " $key = $value ";
}
echo "\n";
```

```
// list properties
echo "properties : ";
$vars =$reflector->getproperties();
foreach ($vars as $ obj)
{
foreach($obj as $a)
{
echo $a->getname() . " ";
}
}
echo "\n";
// list methods
echo "Methods ";
$methods = $reflector->getMethods();
foreach($methods as $obj)
{
echo $obj->getName(). " ";
}
echo "\n";
?>
```

SUMMARY

- A Class is a unit of code composed of variables and functions which describes the characteristics and behaviour of all the members of a set.
- Objects are running instances of the class and contain all the data and information required for the application to function.
- Inheritance is the ability to define a class of one kind as being a sub type of a different kind of class.
- The pseudo-variable *$this* is used when a method is called from within an object context.
- A reference is a variable that "references" the contents of another variable. Passing by reference keeps the target variable "linked" to the source variable, so that if one changes, so does the other.
- Interfaces are a contract between unrelated objects to perform a common function.
- Encapsulation is the ability of an object to protect access to its internal data.

Exercises

A. Answer the following questions: **[1 Mark]**

1. What is use of serialization?
2. How would you declare a class called emptyclass() that has no methods or properties?
3. Which operator is used to access properties and methods of object?
4. What is interface?
5. Which functions are used to notify objects that they are being serialized and unserialised?
6. State the purpose of $this variable.
7. State the purpose of $this variable.

B. Answer the following questions: **[5 Marks]**

1. Explain how to use constructor and destructor to initialize and cleanup your object with suitable example.
2. Write a PHP program to check properties of class and print.
3. Write a PHP script to accept student details (number, name, class) and print it in sorted order of name.
4. Write a PHP program to display information of class and its objects.
5. Write PHP program to create class employee (eno, name, salary, designation) with member functions accept data, display data and max salary. Display data of employees.
6. Write a PHP script to create shape and it's sub-class triangle, square, circle and display the area of selected shape.
7. What is Inheritance? Explain with suitable example.
8. What is an introspection? Explain any four introspective functions provided by PHP.

Questions Asked in Previous Year Exams

1 Mark

1. How to call parent class constructor in PHP. (Write its Syntax)? [Apr. 2019]
2. Which is special function provided by PHP to define constructor? [Apr. 2019]
3. What is serialization? [Apr. 18, Oct. 17, 16]
4. Name any two functions to extract basic information about classes in PHP? [Oct. 17, Apr. 17]
5. Explain the purpose of $ this variable. [Apr. 2017]

4 Marks

1. Write PHP script to demonstrate the concept of introspection for examining object. [Apr. 2019]
2. Write short notes on:
   - i. Serialization [Apr. 2019]
   - ii. Destructor [Apr. 2019]
   - iii. Introspection [Apr. 2018]
   - iv. Constructor [Oct. 17, 16]
3. Define constructor. Explain it with the help of program. [Apr. 18, Oct. 16]
4. Write a PHP script to demonstrate the concept of introspection for examining object. [Apr. 2018]
5. What is introspection? Explain any two introspective function. [Oct. 17, Apr. 17]
6. Define an Interfaces which has methods area(), volume(), define constant PI. Create a class cylinder which implements this (use define()). [Apr. 2017]

7. Define an interface which has methods area(), volume(). Define constant PI. Create a class cylinder which implements this interface and calculate area and volume (use define()). **[Oct. 2016]**

8. How to examine object? Explain it with the help of example. **[Oct. 2016]**