UNIT – 6
CLASS AND OBJECT

UNIVERSITY
OF PETROLEUM
& ENERGY STUDIES

Dr. P S V S Sridhar

Assistant Professor (SS)

Centre for Information Technology

| Jan 2013|

CLASS AND OBJECT

- (1) Introduction, Object, Class
- (2) Defining Class in PHP, Object in PHP
- (3) Usage of $this variable, Constructor, Constructor with Parameters

CREATING CLASSES

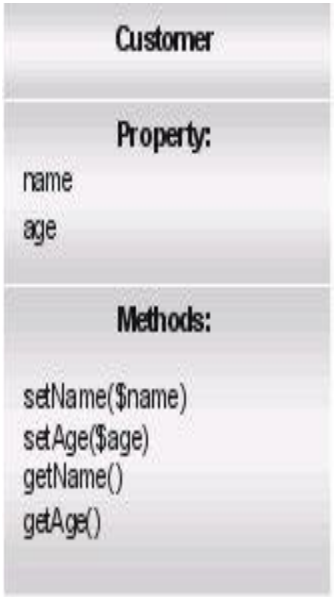
- OOPS starts with classes.
- Classes are the type of objects, in the same way “integer” may be the type of a variable.
- A class is a template that describes attributes and methods of a certain object type.

```
class Classname
{
//Define Properties
//Define Methods
}
```

CREATING CLASSES

```
class Customer
{
private $name;
private $age;

function setName($name)
{
$this->name=$name;
}
function setAge($age)
{
$this->age=$age;
}
function getName()
{
return $this->name;
}
function getAge()
{
return $this->age;
}
}
```



Object in PHP

- Object is a runtime instance of a class.
- An Object is created using the "new" keyword.

  $custobj=new Customer();

- To assign the values for object property

```
$custobj->setName("Joe");
$custobj->setAge(25);
```

Global Variable

```
class Person
{
        var $name;
        function set_name($data)
        {
                global $name;
                $name = $data;
        }
        function get_name()
        {
                global $name;
                return $name;
        }
}
```

It will work, but we won't normally see OOP done using the global keyword. We usually refer the properties of the class using $this keyword. The $this keyword points to the current object.

Usage of $this variable

- $this variable is automatically defined during the execution of the object's method. It is used to refer the properties of an Object.
- The $this keyword points to the current object.

Syntax:

```
$this -> variable_name;
```

You omit the $ in front of the property

```
$this ->name;
```

Setting Access to Properties and Methods

- By default all the members of a class or object are declared **public.**
- You can restrict access to the members of a class or object by using access modifiers
- **public** – Accessible to all.
- **private –** Accessible in the same class.
- **protected –** Accessible in the same class and classes derived from that class.

Public Access

- Public access is the most unrestricted access of all and it is default.

```
<?php
class person
{
        public $name;
        public function set_name($data)
        {
                $this ->name = $data;
        }
}
$obj = new person;
$obj ->set_name(“sridhar”);
?>
```

Private Access

- Private member can't access outside the class or object.

```
<?php
class person
{
        private $name;
        function set_name($data)
        {
                $this ->name = $data;
        }
}
$obj = new person;
$obj ->set_name("sridhar");
?>
```

Member variable is private, you can access from public member function

Member function is private, you can not access from object

Constructor

- Constructors are used to initialize the state of the object during object creation.
- Constructor is a function defined using "__construct" keyword
- Constructor is automatically called during object creation, using the "new" keyword
- Constructor can accept parameters

```
class Customer
{
        private $name;
        private $age;
        function  __construct()
        {
                $this->name="Joe";
                $this->age=25;
        }
        function setName($name)
        {
                $this->name=$name;
        }
        function setAge($age)
        {
                $this->age=$age;
        }
function getName()
{
        return $this->name;
}
function getAge()
{
        return $this->age;
}
}

$custobj=new Customer();
print "Customer Name:".$custobj-
>getName();
print "<br>";
print "Customer Age:".$custobj-
>getAge();
```

Constructor with Parameters

```
function __construct($name,$age)
{
        $this->name=$name;
        $this->age=$age;
}
```

Customer objects are created using the parameterized constructor

```
$custobj1=new Customer("Philip",45);
$custobj2=new Customer("Joe",25);

print "Customer Name:".$custobj1->getName();
print "<br>";
print "Customer Age:".$custobj1->getAge();
print "<br>";
print "Customer Name:".$custobj2->getName();
print “<br>";
print "Customer Age:".$custobj2->getAge();
```

Using Destructors to clean up after objects

- Destructors destroy an object.
- Destructors are named __destruct in PHP(You don't pass arguments to destructor)

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

Inheritance

- **inheritance** is a way to establish Is-a relationship between objects. It is often confused as a way to reuse the existing code which is not a good practice because inheritance for implementation reuse leads to Tight Coupling .

Ex:

```
Class Friend extends Person
{
    …….
}
```

Here we can use the methods of Person class through Friend class. We can also add new methods to the Friend class.

Inheritance

```
class Person
{
        var $name;
        function set_name($data)
        {       $this -> name  = $data;
        }
        function get_name()
        {       return $this -> name;
        }
}
```

```
class Friend  extends Person
{
        var $message;
        function set_message($msg)
        {  $this -> message = $msg;     }
        function speak()
        {   return $this -> message;     }
}

$tony = new Friend;
$tony -> set_name("Tony");
$tony -> set_message("Hi Tony");
echo $tony -> get_name();
echo $tony -> speak();
```

Protected Access

- Protected keyword makes class members accessible only in the class and any class derived from that class.
- <?php
  class MyClass
  {
  public $public = 'Public';
  protected $protected = 'Protected';
  private $private = 'Private';

  function printHello()
  {
  echo $this->public;
  echo $this->protected;
  echo $this->private;
  }
  }

```
$obj = new MyClass();
echo $obj->public; // Works
echo $obj->protected; // Fatal Error
echo $obj->private; // Fatal Error
$obj->printHello(); // Shows Public, Protected and Private
```

Protected Access

```
class MyClass2 extends MyClass
{
protected $protected = 'Protected2';
function printHello()
{
echo $this->public;
echo $this->protected;
echo $this->private;
}
}

$obj2 = new MyClass2();
echo $obj2->public; // Works
echo $obj2->private; // Undefined
echo $obj2->protected; // Fatal Error
$obj2->printHello(); // Shows Public, Protected2, Undefined

?>
```

Constructors and Inheritance

```
class Person
{
        var $name;
        function _construct($data)
        {
                $this ->name =$data;
        }
        function set_name($data)
        {
                $this -> name = $data;
        }
        function get_name()
        {
                return $this -> name;
        }
}
```

```
class Friend extends Person
{
        var $message;
        function _construct($data, $msg)
        {
        parent:: _construct($data);
        $this -> message = $msg;
        }
        function speak()
        {
                echo $this ->message;
        }
}
$nancy = new Friend("Nancy", "Hi");
echo $nancy ->get_name();
echo $nancy ->speak();
?>
```

Overriding Methods

Can redefine a base class method in a derived class.

```
<?php
class Person
{
        var $name;
        function set_name($data)
        {       this->name=$data;       }
        function get_name()
        {       return $this -> name;   }
}
class Friend extends Person
{
        var $name;
        function speak()
        {       echo this->name;        }
        function set_name($data)
        {       $this -> name = strtoupper($data);
        }
}
```

```
$friend = new Friend;
$friend->set_name("susan");
$friend-> speak();
?>
```

Overloading Methods

- Overloading is creating an alternative version with a different argument list.

```
function set_name ($data)
{
        $this -> name = $data;
}
function set_name($data, $msg)
{
        $this -> name = $data;
        $this -> message = $msg;
}
```

```
$friend = new Friend;
$friend ->set_name("susan");
$friend -> set_name("susan", "is here");
```

Overloading Methods

- Previous example will work in standard OOP languages.
- But PHP is different
- Overloading can be implemented in PHP with __call method.
- In a class the original method does not exists __call method will executes

Overloading …

```
<?php
class friend
{
        var $name;
        var $message;
function speak()
{
echo $this->name;
echo $this->message;
}
function set_message($msg)
{
        $this ->message = $msg;
}
function __call($method, $arguments)
{
if ($method == "set_name")
{
if (count($arguments) == 1)
{
$this->name = $arguments[0];
}
if (count($arguments) == 2)
{
$this -> name = $arguments[0];
$this -> message = $arguments[1];
}
}
}
}
$f = new friend;
$f -> set_name("sridhar");
$f -> set_message("hello from sridhar");
$f -> speak();
$f -> set_name("sri", "sri here");
$f -> speak();
?>
```

Set_name function not exists in the example
It will call automatically __call method

Auto loading Classes

- __autoload() function which is automatically called in case you are trying to use a class/interface which hasn't been defined yet.
- This function is passed the names of any classes that PHP is looking for and can’t find in the current file.
- You can load the missing class using require or include
- Where the class class_name is in a file class_name.php

Autoload

person.php

```
<?php
class person
{
        var $name;
        function set_name($data)
        {
                $this -> name = $data;
        }
        function get_data()
        {
                return $this -> name;
        }
}
?>
```

friend.php

```
<?php
class friend extends person
{
        var $message;
        function set_message($msg)
        {
                $this -> message =
$msg;
        }
        function speak()
        {
                echo $this -> message;
        }
}
?>
```

Autoload

```
<?php
function __autoload($class_name)
{
        require $class_name. '.php';
}
$s = new friend;
$s -> set_name("sridhar");
$s -> set_message("hi from sridhar");
echo $s -> get_data();
echo $s-> speak();
?>
```

Thanks