OOP – OBJECT ORIENTED PROGRAMMING

INTRODUCTION

Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure.

In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects.

CLASS

In object-oriented programming , a class is a template definition of the method s and variable s in a particular kind of object . Thus, an object is a specific instance of a class; it contains real values instead of variables.

The class is one of the defining ideas of object-oriented programming. Among the important ideas about classes are:

A class can have subclasses that can inherit all or some of the characteristics of the class. In relation to each subclass, the class becomes the superclass. Subclasses can also define their own methods and variables that are not part of their superclass. The structure of a class and its subclasses is called the class hierarchy.

PROPERTY

Properties are an object-oriented idiom. The term describes a one or two functions (depending on the desired program behavior) - a 'getter' that retrieves a value and a 'setter' that sets a value. By convention, properties usually don't have many side-effects. (And the side-effects they do have usually have limited scope: they may validate the item being set, notify listeners of a change, or convert an object's private data to or from a publicly-declared type.)

METHOD

A method in object-oriented programming is a procedure associated with a class. A method defines the behavior of the objects that are created from the class. Another way to say this is that a method is an action that an object is able to perform. The association between method and class is called binding. Consider the example of an object of the type 'person,' created using the person class. Methods associated with this class could consist of things like walking and driving. Methods are sometimes confused with functions, but they are distinct.

```
<?php
class SimpleClass{
    // property declaration
    public $var = 'a default value';
```

```
    // method declaration
    public function displayVar() {
        echo $this->var;
    }
}
?>
```

VISIBILITY

There are 3 type of visibility available in php for controlling your property or method.

- **Public**: Public method or variable can be accessible from anywher. I mean from inside the class, out side the class and in child(will dicuss in next chapter) class also.
- **Private**: Method or property with private visibility can only be accessible inside the class. You can not access private method or variable from outside of your class.
- **Protected**: Method or variable with protected visibility can only be access in the derived class. Or in other word in child class. Protected will be used in the process of inheritance.

PUBLIC VISIBILITY IN PHP CLASSES

Public visiblity is least restricted visibility available in php. If you will not define the visibity factor with your method or property then public will be by defautl applied. Public methods or variables can be accessible from anywhere.For example, It can be accessible from using object(outside the class), or inside the class, or in child class. Following is the example of the public visibility in php classes:

```
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
```

PRIVATE VISIBILITY IN PHP CLASSES

Private method or properties can only be accessible withing the class. You can not access private variable or function of the class by making object out side the class. But you can use private function and property within the class using $this object. Private visibility in php classes is used when you do not want your property or function to be exposed outside the class.

```
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;//Throw fatal error of visibility
$objT->pubDo("test");//Print "test"
$objT->privDo(1);//Fatal error of visibility
$objT->pubPrivDo();//Within this method private function privDo and
variable xyz is called using $this variable.
```

PROTECTED VISIBILITY IN PHP CLASSES

Protected visibility in php classes are only useful in case of inheritance and interface. We will discuss in dept of interfaces and inheritance in other chapter of this tutorial. Protected method or variable can be accessible either within class or child class.

```
class parent{
      protected $pr;
      public $a
      protected function testParent(){
            echo this is test;
      }
}
class child extends parent{
      public function testChild(){
            $this->testParent(); //will work because it
      }
}
$objParent = new parent();
$objParent->testParent();//Throw error
$objChild = new Child();
$objChild->setChild();//work because test child will call test
parent.
```

CONSTRUCTOR

ll objects can have a special built-in method called a 'constructor'. Constructors allow you to initialise your object's properties (translation: give your properties values,) when you instantiate (create) an object.

Note: If you create a __construct() function (it is your choice,) PHP will automatically call the __construct() method/function when you create an object from your class. The 'construct' method starts with two underscores (__) and the word 'construct'.

```
class person {
      var $name;
      function __construct($persons_name) {
            $this->name = $persons_name;
      }

      function set_name($new_name) {
             $this->name = $new_name;
      }

      function get_name() {
             return $this->name;
       }

}
```

DESTRUCTOR

The Destructor method will be called as soon as there are no other references to a particular object, or in any order during the shutdown sequence. Destructor automatically call at last.

```
class demo{
      function  __construct(){
            echo  "object is initializing their propertie"."<br/>";
      }
      function work(){
            echo  "Now works is going "."<br/>";
      }
      function __destruct(){
            echo  "after completion the work,  object  destroyed
automatically";
      }
}
$obj= new demo();
$obj->work();
echo is_object($obj);
```

INHERITANCE

One of the main advantages of object-oriented programming is the ability to reduce code duplication with inheritance. Code duplication occurs when a programmer writes the same code more than once, a problem that inheritance strives to solve. In inheritance, we have a parent class with its own methods and properties, and a child class (or classes) that can use the code from the

parent. By using inheritance, we can create a reusable piece of code that we write only once in the parent class, and use again as much as we need in the child classes.

```
//The parent class
class Car {
  private $model;
  public function setModel($model){
    $this -> model = $model;
  }
  public function hello(){
    return "beep! I am a <i>" . $this -> model . "</i><br />";
  }
}

class SportsCar extends Car {
  //No code in the child class
}

$sportsCar1 = new SportsCar();
$sportsCar1 -> setModel('Mercedes Benz');
echo $sportsCar1 -> hello();
```

SCOPE RESOLUTION OPERATOR (::)

The Scope Resolution Operator (also called Paamayim Nekudotayim) or in simpler terms, the double colon, is a token that allows access to static, constant, and overridden properties or methods of a class.

When referencing these items from outside the class definition, use the name of the class. As of PHP 5.3.0, it's possible to reference the class using a variable. The variable's value can not be a keyword (e.g. self, parent and static).

```
//from outside the class definition
class MyClass {
    const CONST_VALUE = 'A constant value';
}

$classname = 'MyClass';
echo $classname::CONST_VALUE; // As of PHP 5.3.0
echo MyClass::CONST_VALUE;
//from inside the class definition
class OtherClass extends MyClass{
    public static $my_static = 'static var';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
```

```
}
$classname = 'OtherClass';
$classname::doubleColon(); // As of PHP 5.3.0
OtherClass::doubleColon();
```

AUTOLOADING CLASSES

When the number of classes grows, it is quite tedious to call require_once() function everywhere you use the classes. Fortunately, PHP provides a mechanism that allows you to load class files automatically whenever you try to create an object from a non-existent class in the context of the current script file.

PHP will call __autoload() function automatically whenever you create an object from a non-existent class. To load class files automatically, you need to implement the __autoload() function somewhere in your web application.

```
function __autoload($className) {
      $className = str_replace('..', '', $className);
      require_once("classes/$className.php");
}
```

Although the __autoload() function can also be used for autoloading classes and interfaces, it's preferred to use the spl_autoload_register() function. This is because it is a more flexible alternative (enabling for any number of autoloaders to be specified in the application, such as in third party libraries). For this reason, using __autoload() is discouraged and it may be deprecated in the future.

```
spl_autoload_register(function ($class_name) {
    include $class_name . '.php';
});

$obj  = new MyClass1();
$obj2 = new MyClass2();
```

CLASS CONSTANTS

It is possible to define constant values on a per-class basis remaining the same and unchangeable. Constants differ from normal variables in that you don't use the $ symbol to declare or use them. The default visibility of class constants is public.

The value must be a constant expression, not (for example) a variable, a property, or a function call. It's also possible for interfaces to have constants. Look at the interface documentation for examples. As of PHP 5.3.0, it's possible to reference the class using a variable. The variable's value can not be a keyword (e.g. self, parent and static). Note that class constants are allocated once per class, and not for each class instance.

```
class MyClass{
    const CONSTANT = 'constant value';
```

```
    function showConstant() {
        echo  self::CONSTANT . "\n";
    }
}
echo MyClass::CONSTANT . "<br />";
$classname = "MyClass";
echo $classname::CONSTANT . "<br />"; // As of PHP 5.3.0
$class = new MyClass();
$class->showConstant();
echo $class::CONSTANT."<br />"; // As of PHP 5.3.0
```

MYSQL DATABASE HANDLING WITH OOP

```
//dbconfig
<?php
define('DB_SERVER','localhost');
define('DB_USER','root');
define('DB_PASSWORD','');
define('DB_NAME','oop');
class DB_con{
     function __construct()
     {
          $conn = mysql_connect(DB_SERVER,DB_USER,DB_PASSWORD) or
die('error connecting to server'.mysql_error());
          mysql_select_db(DB_NAME, $conn) or die('error connecting
to database->'.mysql_error());
     }
}
?>
```

```
//class.crud
<?php
include_once 'dbconfig.php';
class CRUD{
     public function __construct(){
          $db = new DB_con();
     }

     // function for create
     public function create($fname,$lname,$city) {
          mysql_query("INSERT INTO
users(first_name,last_name,user_city)
VALUES('$fname','$lname','$city')");
     }

     // function for read
     public function read(){
```

```
            return mysql_query("SELECT * FROM users ORDER BY user_id
ASC");
      }

      // function for delete
      public function delete($id) {
            mysql_query("DELETE FROM users WHERE user_id=".$id);
      }

      // function for update
      public function update($fname,$lname,$city,$id) {
            mysql_query("UPDATE users SET first_name='$fname',
last_name='$lname', user_city='$city' WHERE user_id=".$id);
      }
}
?>
```

```
//dbcrud
<?php
include_once 'class.crud.php';
$crud = new CRUD();
if(isset($_POST['save'])) {
      $fname = $_POST['fname'];
      $lname = $_POST['lname'];
      $city = $_POST['city'];

      // insert
     $crud->create($fname,$lname,$city);
      // insert
      header("Location: index.php");
}
if(isset($_GET['del_id'])) {
      $id = $_GET['del_id'];
      $crud->delete($id);
      header("Location: index.php");
}
if(isset($_POST['update'])) {
      $id = $_GET['edt_id'];
      $fname = $_POST['fname'];
      $lname = $_POST['lname'];
      $city = $_POST['city'];
      $crud->update($fname,$lname,$city,$id);
      header("Location: index.php");
}
?>
```

```
//add_records
<!DOCTYPE html5>
<html>
    <head>
    <link rel="stylesheet" href="style.css" type="text/css" />
    </head>
    <body>
        <div id="header">
            <label>php oops crud tutorial part-2 by
cleartuts</label>
        </div>
        <center>
            <form method="post" action="dbcrud.php">
                <table id="dataview">
                <tr>
                    <td><input type="text" name="fname"
placeholder="first name" /></td>
                </tr>
                <tr>
                    <td><input type="text" name="lname"
placeholder="last name" /></td>
                </tr>
                <tr>
                    <td><input type="text" name="city"
placeholder="city" /></td>
                </tr>
                <tr>
                    <td><button type="submit"
name="save">save</button></td>
                </tr>
                </table>
            </form>
        </center>
    </body>
</html>
```

```
//edit_records
<?php
    include_once 'class.crud.php';    $crud = new CRUD();
    if(isset($_GET['edt_id'])){
        $res=mysql_query("SELECT * FROM users WHERE
user_id=".$_GET['edt_id']);
        $row=mysql_fetch_array($res);
    }
?>
<!DOCTYPE html5>
```

```
<html>
    <head>
        <link rel="stylesheet" href="style.css" type="text/css" />
    </head>
    <body>
        <div id="header">
            <label>php oops crud tutorial part-2 by
cleartuts</label>
        </div>
        <center>
            <form method="post" action="dbcrud.php?edt_id=<?php echo
$_GET['edt_id'] ?>">
                <table id="dataview">
                    <tr><td><input type="text" name="fname"
placeholder="first name" value="<?php echo $row['first_name'] ?>"
/><br /></td></tr>
                    <tr><td><input type="text" name="lname"
placeholder="last name" value="<?php echo $row['last_name'] ?>"
/></td></tr>
                    <tr><td><input type="text" name="city"
placeholder="city" value="<?php echo $row['user_city'] ?>"
/></td></tr>
                    <tr><td><button type="submit"
name="update">update</button></td></tr>
                </table>
            </form>
        </center>
    </body>
</html>
```

```
//index
<?php
    include_once 'class.crud.php';    $crud = new CRUD();
?>
<html>
    <head>
        <link rel="stylesheet" href="style.css" type="text/css" />
    </head>
    <body>
        <div id="header">
            <label>php oops crud tutorial part-2 by
cleartuts</label>
        </div>
        <center>
            <table id="dataview">
                <tr>
```

```
                    <td colspan="5"><a href="add_records.php">add
new</a></td>
                </tr>
                <?php
                    $res = $crud->read();
                    if(mysql_num_rows($res)>0)
                    {
                        while($row = mysql_fetch_array($res))
                        {
                ?>
                <tr>
                <td><?php echo $row['first_name']; ?></td>
                <td><?php echo $row['last_name']; ?></td>
                <td><?php echo $row['user_city']; ?></td>
                <td><a href="edit_records.php?edt_id=<?php echo
$row['user_id']; ?>">edit</a></td>
                <td><a href="dbcrud.php?del_id=<?php echo
$row['user_id']; ?>">delete</a></td>
                </tr>
                <?php
                        }
                    }
                    else
                    {
                ?>
                <tr><td colspan="5">Nothing here... add some
new</td></tr>
                <?php
                    }
                ?>
            </table>
        </center>
    </body>
</html>
```

```
//style
@charset "utf-8";
* { margin:0; padding:0; }
#header{
     text-align:center;
     width:100%;
     height:50px;
     background:#00a2d1;
     color:#f9f9f9;
     font-weight:bolder;
     font-family:Verdana, Geneva, sans-serif;
```

```
      font-size:35px;
}
table,td{
      width:40%;
      padding:15px;
      border:solid #e1e1e1 1px;
      font-family:Verdana, Geneva, sans-serif;
      border-collapse:collapse;
}
#dataview{
      margin-top:100px;
      position:relative;
      bottom:50px;
}
#dataview input{
      width:100%;
      height:40px;
      border:0; outline:0;
      font-family:Verdana, Geneva, sans-serif;
      padding-left:10px;
}
#dataview button{
      width:200px;
      height:40px;
      border:0; outline:0;
      font-family:Verdana, Geneva, sans-serif;
      padding-left:10px;
}
footer { margin-top:50px; position:relative; bottom:50px; font-
family:Verdana, Geneva, sans-serif; }
```

```
//SQL
CREATE TABLE IF NOT EXISTS `users` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(25) NOT NULL,
  `last_name` varchar(25) NOT NULL,
  `user_city` varchar(50) NOT NULL,
  PRIMARY KEY (`user_id`)
)
```