29. PHP – Object Oriented Programming

We can imagine our universe made of different objects like sun, earth, moon etc. Similarly, we can imagine our car made of different objects like wheel, steering, gear etc. In the same way, there are object oriented programming concepts which assume everything as an object and implement a software using different objects.

Object Oriented Concepts

Before we go in detail, let's define important terms related to Object Oriented Programming.

- **Class:** This is a programmer-defined datatype, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.

- **Object:** An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as **instance**.

- **Member Variable:** These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called **attribute** of the object once an object is created.

- **Member function:** These are the function defined inside a class and are used to access object data.

- **Inheritance:** When a class is defined by inheriting existing function of a parent class, then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.

- **Parent class:** A class that is inherited from by another class. This is also called a base class or super class.

- **Child Class:** A class that inherits from another class. This is also called a subclass or derived class.

- **Polymorphism:** This is an object oriented concept where the same function can be used for different purposes. For example, function name will remain same but it may take different number of arguments and can do different task.

- **Overloading:** a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly, functions can also be overloaded with different implementation.

- **Data Abstraction:** Any representation of data in which the implementation details are hidden (abstracted).

- **Encapsulation:** refers to a concept where we encapsulate all the data and member functions together to form an object.

- **Constructor:** refers to a special type of function which will be called automatically whenever there is an object formation from a class.

- **Destructors:** refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

Defining PHP Classes

The general form for defining a new class in PHP is as follows:

```
<?php
class phpClass{
   var $var1;
   var $var2 = "constant string";
   function myfunc ($arg1, $arg2) {
      [..]
   }
   [..]
}
?>
```

Here is the description of each line:

- The special form **class**, followed by the name of the class that you want to define.

- A set of braces enclosing any number of variable declarations and function definitions.

- Variable declarations start with the special form **var**, which is followed by a conventional $ variable name; they may also have an initial assignment to a constant value.

- Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.

Example

Here is an example which defines a class of Books type:

```
<?php
class  Books{
    /* Member variables */
    var $price;
    var $title;
    /* Member functions */
    function setPrice($par){
       $this->price = $par;
    }
```

```
    function getPrice(){
       echo $this->price ."<br/>";
    }
    function setTitle($par){
       $this->title = $par;
    }
    function getTitle(){
       echo $this->title ." <br/>";
    }
}
?>
```

The variable **$this** is a special variable and it refers to the same object, i.e., itself.

Creating Objects in PHP

Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using **new** operator.

```
$physics = new Books;
$maths = new Books;
$chemistry = new Books;
```

Here we have created three objects and these objects are independent of each other and they will have their existence separately. Next, we will see how to access member function and process member variables.

Calling Member Functions

After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only. Following example shows how to set title and prices for the three books by calling member functions.

```
$physics->setTitle( "Physics for High School" );
$chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );

$physics->setPrice( 10 );
$chemistry->setPrice( 15 );
$maths->setPrice( 7 );
```

Now you call another member functions to get the values set by in above example:

```
   $physics->getTitle();
   $chemistry->getTitle();
   $maths->getTitle();
   $physics->getPrice();
   $chemistry->getPrice();
   $maths->getPrice();
```

This will produce the following result:

```
  Physics for High School
  Advanced Chemistry
  Algebra
  10
  15
  7
```

Constructor Functions

Constructor Functions are special type of functions which are called automatically whenever an object is created. So we take full advantage of this behavior, by initializing many things through constructor functions.

PHP provides a special function called **__construct()** to define a constructor. You can pass as many as arguments you like into the constructor function.

Following example will create one constructor for Books class and it will initialize price and title for the book at the time of object creation.

```
function __construct( $par1, $par2 ){
   $this->price = $par1;
   $this->title = $par2;
}
```

Now we don't need to call set function separately to set price and title. We can initialize these two member variables at the time of object creation only. Check following example below:

```
   $physics = new Books( "Physics for High School", 10 );
   $maths = new Books ( "Advanced Chemistry", 15 );
   $chemistry = new Books ("Algebra", 7 );

   /* Get those set values */
   $physics->getTitle();
   $chemistry->getTitle();
```

```
   $maths->getTitle();

   $physics->getPrice();
   $chemistry->getPrice();
   $maths->getPrice();
```

This will produce the following result:

```
  Physics for High School
  Advanced Chemistry
  Algebra
  10
  15
  7
```

Destructor

Like a constructor function you can define a destructor function using function **__destruct()**. You can release all the resources with-in a destructor.

Inheritance

PHP class definitions can optionally inherit from a parent class definition by using the extends clause. The syntax is as follows:

```
  class Child extends Parent {
     <definition body>
  }
```

The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics:

- Automatically has all the member variable declarations of the parent class.
- Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.

Following example inherit Books class and adds more functionality based on the requirement.

```
class Novel extends Books{
   var publisher;
   function setPublisher($par){
     $this->publisher = $par;
   }
```

```
   function getPublisher(){
     echo $this->publisher. "<br />";
   }
}
```

Now apart from inherited functions, class Novel keeps two additional member functions.

Function Overriding

Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.
In the following example, getPrice and getTitle functions are overriden to retrun some values.

```
    function getPrice(){
       echo $this->price . "<br/>";
       return $this->price;
    }
    function getTitle(){
       echo $this->title . "<br/>";
       return $this->title;
    }
```

Public Members

Unless you specify otherwise, properties and methods of a class are public. That is to say, they may be accessed in three possible situations:

- From outside the class in which it is declared
- From within the class in which it is declared
- From within another class that implements the class in which it is declared

Till now we have seen all members as public members. If you wish to limit the accessibility of the members of a class, then you define class members as **private** or **protected**.

Private members

By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.

A class member can be made private by using **private** keyword in front of the member.

```
class MyClass {
```

```
   private $car = "skoda";
   $driver = "SRK";

   function __construct($par) {
      // Statements here run every time an instance of the class is created.
   }

   function myPublicFunction() {
      return("I'm visible!");
   }
   private function myPrivateFunction() {
      return("I'm  not visible outside!");
   }
}
```

When *MyClass* class is inherited by another class using extends, myPublicFunction() will be visible, as will $driver. The extending class will not have any awareness of or access to myPrivateFunction and $car, because they are declared private.

Protected members

A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class. Protected members are not available outside of those two kinds of classes. A class member can be made protected by using **protected** keyword in front of the member.

Here is different version of MyClass:

```
class MyClass {
   protected $car = "skoda";
   $driver = "SRK";

   function __construct($par) {
      // Statements here run every time
      // an instance of the class
      // is created.
   }
   function myPublicFunction() {
      return("I'm visible!");
   }
   protected function myPrivateFunction() {
      return("I'm  visible in child class!");
```

```
   }
}
```

Interfaces

Interfaces are defined to provide a common function names to the implementors. Different implementors can implement those interfaces according to their requirements. You can say, interfaces are skeletons which are implemented by developers.

As of PHP5, it is possible to define an interface, like this:

```
interface Mail {
   public function sendMail();
}
```

Then, if another class implemented that interface, like this:

```
class Report implements Mail {
   // sendMail() Definition goes here
}
```

Constants

A constant is somewhat like a variable, in that it holds a value, but is really more like a function because a constant is immutable. Once you declare a constant, it does not change. Declaring one constant is easy, as is done in this version of MyClass:

```
class MyClass {
   const requiredMargin = 1.7;
   function __construct($incomingValue) {
      // Statements here run every time
      // an instance of the class
      // is created.
   }
}
```

In this class, requiredMargin is a constant. It is declared with the keyword const, and under no circumstances can it be changed to anything other than 1.7. Note that the constant's name does not have a leading $, as variable names do.

Abstract Classes

An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword **abstract**, like this:

When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.

```
abstract class MyAbstractClass {
   abstract function myAbstractFunction() {
   }
}
```

Note that the function definitions inside an abstract class must also be preceded by the keyword abstract. It is not legal to have abstract function definitions inside a non-abstract class.

Static Keyword

Declaring class members or methods as static makes them accessible without needing an instantiation of the class. A member declared as static cannot be accessed with an instantiated class object (though a static method can).

Try out the following example:

```
<?php
class Foo
{
    public static $my_static = 'foo';

    public function staticValue() {
        return self::$my_static;
    }
}
print Foo::$my_static . "\n";
$foo = new Foo();
print $foo->staticValue() . "\n";
```

Final Keyword

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final, then it cannot be extended.

The following example results in Fatal error: Cannot override final method BaseClass::moreTesting()

```
<?php
class BaseClass {
   public function test() {
```

```
        echo "BaseClass::test() called<br>";
    }

    final public function moreTesting() {
        echo "BaseClass::moreTesting() called<br>";
    }
}

class ChildClass extends BaseClass {
    public function moreTesting() {
        echo "ChildClass::moreTesting() called<br>";
    }
}
?>
```

Calling parent constructors

Instead of writing an entirely new constructor for the subclass, let's write it by calling the parent's constructor explicitly and then doing whatever is necessary in addition for instantiation of the subclass. Here's a simple example:

```
class Name
{
   var $_firstName;
   var $_lastName;
   function Name($first_name, $last_name)
   {
     $this->_firstName = $first_name;
     $this->_lastName = $last_name;
   }
   function toString() {
     return($this->_lastName ."," .$this->_firstName);
   }
}
class NameSub1 extends Name
{
   var $_middleInitial;
```

```
    function NameSub1($first_name, $middle_initial, $last_name) {
        Name::Name($first_name, $last_name);
        $this->_middleInitial = $middle_initial;
    }


    function toString() {
        return(Name::toString() . " " . $this->_middleInitial);
    }
}
```

In this example, we have a parent class (Name), which has a two-argument constructor, and a subclass (NameSub1), which has a three-argument constructor. The constructor of NameSub1 functions by calling its parent constructor explicitly using the :: syntax (passing two of its arguments along) and then setting an additional field. Similarly, NameSub1 defines its nonconstructor toString() function in terms of the parent function that it overrides.

NOTE: A constructor can be defined with the same name as the name of a class. It is defined in above example.

30. PHP – PHP for C Developers

The simplest way to think of PHP is as interpreted C that you can embed in HTML documents. The language itself is a lot like C, except with untyped variables, a whole lot of Web-specific libraries built in, and everything hooked up directly to your favorite Web server.
The syntax of statements and function definitions should be familiar, except that variables are always preceded by $, and functions do not require separate prototypes.
Here we will put some similarities and differences in PHP and C

Similarities

- **Syntax:** Broadly speaking, PHP syntax is the same as in C: Code is blank insensitive, statements are terminated with semicolons, function calls have the same structure (my_function(expression1, expression2)), and curly braces ({ and }) make statements into blocks. PHP supports C and C++-style comments (/* */ as well as //), and also Perl and shell-script style (#).

- **Operators:** The assignment operators (=, +=, *=, and so on), the Boolean operators (&&, ||, !), the comparison operators (<,>, <=, >=, ==, !=), and the basic arithmetic operators (+, -, *, /, %) all behave in PHP as they do in C.

- **Control structures:** The basic control structures (if, switch, while, for) behave as they do in C, including supporting break and continue. One notable difference is that switch in PHP can accept strings as case identifiers.

- **Function names:** As you peruse the documentation, you.ll see many function names that seem identical to C functions.

Differences

- **Dollar signs:** All variables are denoted with a leading $. Variables do not need to be declared in advance of assignment, and they have no intrinsic type.

- **Types:** PHP has only two numerical types: integer (corresponding to a long in C) and double (corresponding to a double in C). Strings are of arbitrary length. There is no separate character type.

- **Type conversion:** Types are not checked at compile time, and type errors do not typically occur at runtime either. Instead, variables and values are automatically converted across types as needed.

- **Arrays:** Arrays have a syntax superficially similar to C's array syntax, but they are implemented completely differently. They are actually associative arrays or hashes, and the index can be either a number or a string. They do not need to be declared or allocated in advance.

- **No structure type:** There is no struct in PHP, partly because the array and object types together make it unnecessary. The elements of a PHP array need not be of a consistent type.

- **No pointers:** There are no pointers available in PHP, although the typeless variables play a similar role. PHP does support variable references. You can also emulate function pointers to some extent, in that function names can be stored in variables and called by using the variable rather than a literal name.

- **No prototypes:** Functions do not need to be declared before their implementation is defined, as long as the function definition can be found somewhere in the current code file or included files.

- **Memory management:** The PHP engine is effectively a garbage-collected environment (reference-counted), and in small scripts there is no need to do any deallocation. You should freely allocate new structures - such as new strings and object instances. IN PHP5, it is possible to define destructors for objects, but there is no free or delete. Destructors are called when the last reference to an object goes away, before the memory is reclaimed.

- **Compilation and linking:** There is no separate compilation step for PHP scripts.

- **Permissiveness:** As a general matter, PHP is more forgiving than C (especially in its type system) and so will let you get away with new kinds of mistakes. Unexpected results are more common than errors.