Object Oriented Programming

Key topics covered

1. Software Crisis
2. Software Evaluation
3. Procedure-Oriented Programming
4. Object Oriented Paradigm
5. Object-oriented concepts
    a. Objects
    b. Classes
    c. Data Abstraction and Data Encapsulation
    d. Inheritance
    e. Polymorphism
6. Creating Classes, Attributes, and Operations in PHP
7. Constructors
8. Destructors
9. Instantiating Classes
10. Using Class Attributes
11. Controlling Access with private and public
12. Calling Class Operations
13. Controlling Visibility Through Inheritance with private and protected
14. Overriding
15. Preventing Inheritance and Overriding with final
16. Implementing Interfaces
17. Using Per-Class Constants
18. Implementing Static Methods
19. Checking Class Type and Type Hinting
20. Late Static Bindings
21. Cloning Objects
22. Using Abstract Classes
23. Overloading Methods with __call()
24. Using __autoload()
25. Implementing Iterators and Iteration
26. Converting Your Classes to Strings
27. Using the Reflection API
28. Classes/Object Functions

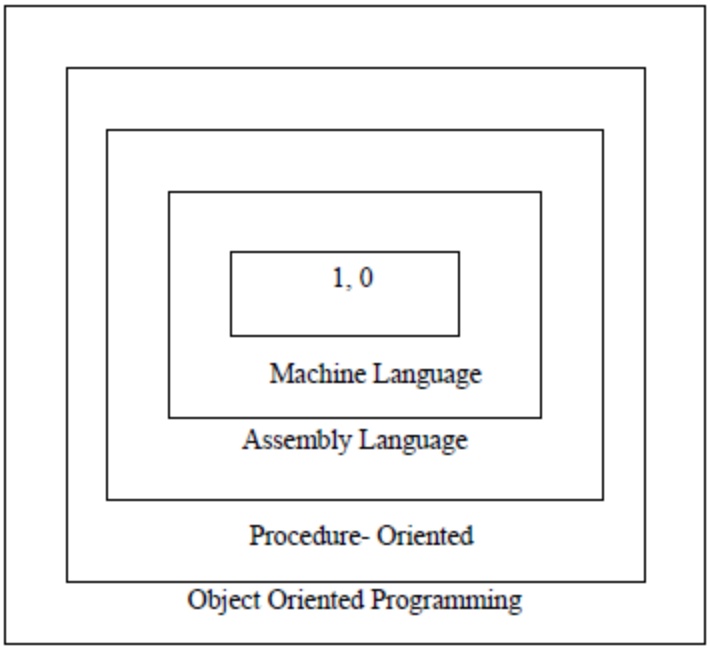
1. Software Crisis

Developments in software technology continue to be dynamic. New tools and techniques are announced in quick succession. This has forced the software engineers and industry to continuously look for new approaches to software design and development, and they are becoming more and more critical in view of the increasing complexity of software systems as well as the highly competitive nature of the industry. These rapid advances appear to have created a situation of crisis within the industry.

The following issues need to be addressed to face the crisis:

- How to represent real-life entities of problems in system design?
- How to design system with open interfaces?
- How to ensure reusability and extensibility of modules?
- How to develop modules that are tolerant of any changes in future?
- How to improve software productivity and decrease software cost?
- How to improve the quality of software?
- How to manage time schedules?

2. Software Evaluation



3. Procedure-Oriented Programming

In the procedure oriented approach, the problem is viewed as the sequence of things to be done such as reading, calculating and printing such as cobol, fortran and c. The primary focus is on functions. A typical structure for procedural programming is shown as below

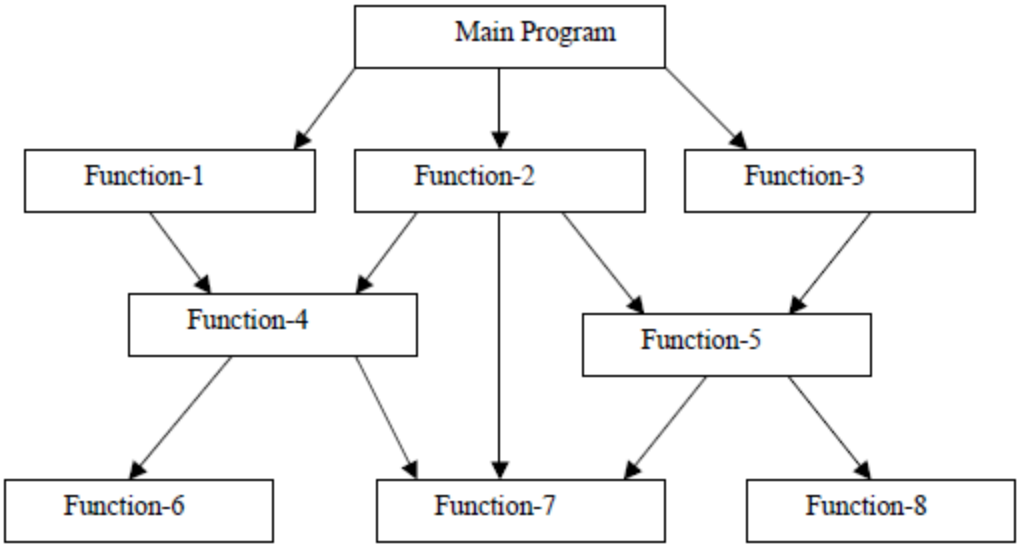


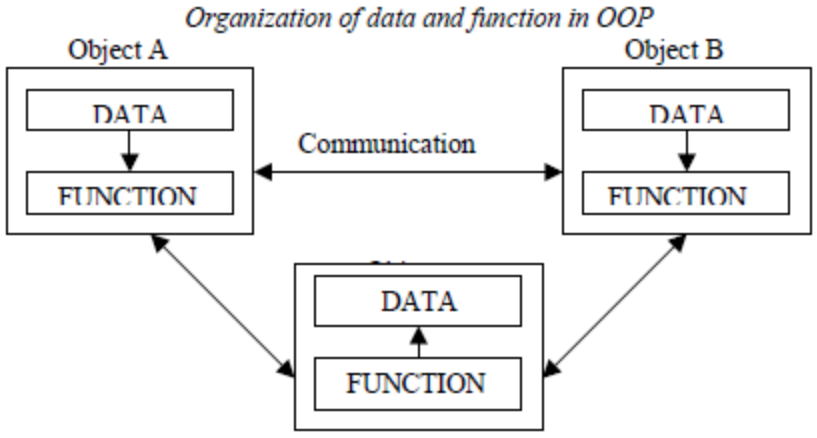
Fig. 1.2 Typical structure of procedural oriented programs

4. Object Oriented Paradigm

The major motivating factor in the invention of object-oriented approach is to remove some of the flaws encountered in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the function that operate on it, and protects it from accidental modification from outside function.

OOP allows decomposition of a problem into a number of entities called objects and then builds data and function around these objects. The organization of data and function in object-oriented programs is shown in figure.

The data of an object can be accessed only by the function associated with that object. However, function of one object can access the function of other objects.



Some of the features of object oriented programming are:

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are ties together in the data structure.
- Data is hidden and cannot be accessed by external function.
- Objects may communicate with each other through function.
- New data and functions can be easily added whenever necessary.
- Follows bottom up approach in program design.

Object-oriented programming is the most recent concept among programming paradigms and still means different things to different people.

5. Object-Oriented Concepts

Modern programming languages usually support or even require an object-oriented approach to software development. Object-oriented development attempts to use the classifications, relationships, and properties of the objects in the system to aid in program development and code reuse.

Object-oriented software's central advantage is its capability to support and encourage **encapsulation**—also known as **data hiding**. Essentially, access to the data within an object is available only via the object's operations, known as the **interface** of the object.

Object orientation can help you to manage the complexity in your projects, increase code reusability, and thereby reduce maintenance costs.

You can think of the noun bicycle as a class of objects describing many distinct bicycles with many common features or attributes—
Such as two wheels, a color, and a size—and operations, such as move.

My own bicycle can be thought of as an object that fits into the class bicycle. It has all the common features of all bicycles, including a move operation that behaves the same as most other bicycles' move—even if it is used more rarely. My bicycle's attributes have unique values because my bicycle is green, and not all bicycles are that color.

Example:

```
class vehicle{
   public $color = "black";
   public $wheels = "four";
   public $size = "medium";
   function move($speed){
      echo $speed;
   }
}

class car extends vehicle{

   public $color = "red";
   public $wheels = "four";
   public $size = "medium";
   function move($speed) {
      echo $speed;
```

```
    }
  }

  class bicycle extends car{
     public $color = "green";
     public $wheels = "two";
     public $size = "small";
     function move($speed){
        echo $speed;
    }
  }
```

a. Objects

Objects are the basic run time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.

They may also represent user-defined data such as vectors, time and lists. Programming problem is analyzed in term of objects and the nature of communication between them. Program objects should be chosen such that they match closely with the real-world objects. Objects take up space in the memory and have an associated address like a record in Pascal, or a structure in c.

When a program is executed, the objects interact by sending messages to one another. For example, if "customer" and "account" are to object in a program, then the customer object may send a message to the count object requesting for the bank balance. Each object contain data, and code to manipulate data. Objects can interact without having to know details of each other's data or code. It is a sufficient to know the type of message accepted, and the type of response returned by the objects. Although different author represent them differently fig. shows two notations that are popularly used in object-oriented analysis and design.

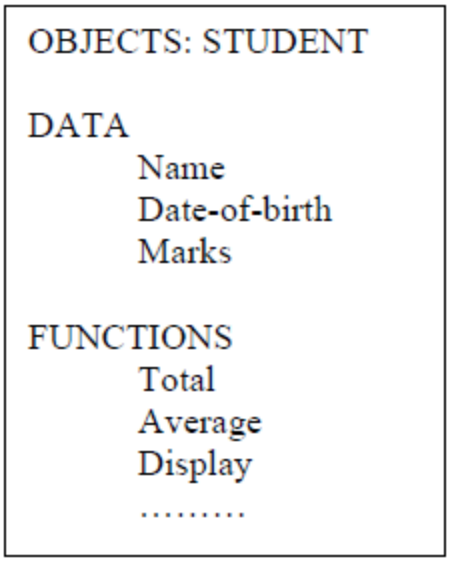


Fig. 1.5 representing an object

b. Classes

We just mentioned that objects contain data, and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of class.

In fact, objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus a collection of objects similar types.

For example: Mango, Apple and orange members of class fruit.

Classes are user-defined that types and behave like the built-in types of a programming language. The syntax used to create an object is not different then the syntax used to create an integer object in C. If fruit has been defines as a class, then the statement

Fruit Mango;
Will create an object mango belonging to the class fruit.

c. Data Abstraction and Data Encapsulation

Encapsulate means to hide. Encapsulation is also called data hiding. You can think Encapsulation like a capsule (medicine tablet) which hides medicine inside it. Encapsulation is wrapping, just hiding properties and methods. Encapsulation is used for hide the code and data in a single unit to protect the data from the outside the world. Class is the best example of encapsulation.

Abstraction refers to showing only the necessary details to the intended user. As the name suggests, abstraction is the "abstract form of anything". We use abstraction in programming languages to make abstract class. Abstract class represents abstract view of methods and properties of class.

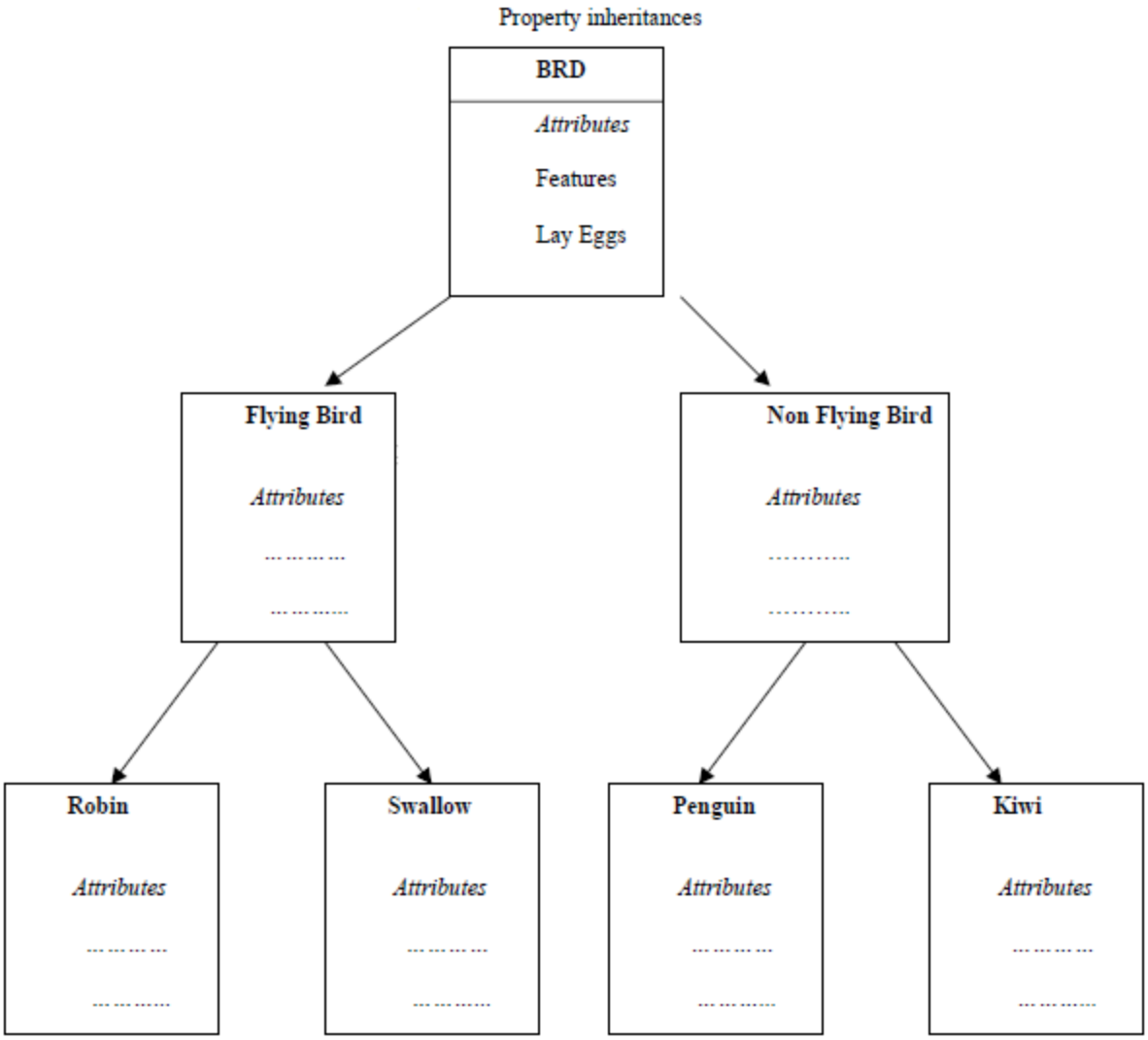
d. Polymorphism

An object-oriented programming language must support polymorphism, which means that different classes can have different behaviors for the same operation.

For Example: - you have a class car and a class bicycle, both can have different move operations. For real world objects, this would rarely be a problem. Bicycles are not likely to become confused and start using a car's move operation instead. However, a programming language does not possess the common sense of the real world, so the language must support **polymorphism** to know which move operation to use on a particular object.

e. Inheritance

Inheritance is the process by which objects of one class acquired the properties of objects of another classes. It supports the concept of hierarchical classification. For example, the bird, 'robin' is a part of class 'flying bird' which is again a part of the class 'bird'. The principal behind this sort of division is that each derived class shares common characteristics with the class from which it is derived as illustrated in fig.
In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined feature of both the classes. The real appeal and power of the inheritance mechanism is that it



Allows the programmer to reuse a class i.e almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduced any undesirable side-effects into the rest of classes.

Inheritance allows you to create a hierarchical relationship between classes using subclasses.
A subclass inherits attributes and operations from its superclass.

For example, car and bicycle have some things in common. You could use a class vehicle to contain the things such as a color attribute and a move operation that all vehicles have, and then let the car and bicycle classes inherit from vehicle.

You will hear subclass, derived class, and child used interchangeably. Similarly, you will hear superclass and parent used interchangeably.

Implementing Inheritance in PHP

```
class B extends A{
        public $attribute2;
        function operation2(){
        }
}
```

If the class A was declared as

```
class A{
```

```
        public $attribute1;
        function operation1(){
        }
}
```

All the following accesses to operations and attributes of an object of type B would be valid:

```
$b = new B();
$b->operation1();
$b->attribute1 = 10;
$b->operation2();
$b->attribute2 = 10;
```

It is important to note that inheritance works in only one direction. The subclass or child inherits features from its parent or super class, but the parent does not take on features of the child. This means that the last two lines in this code are wrong:

```
$a = new A();
$a->operation1();
$a->attribute1 = 10;
$a->operation2();
$a->attribute2 = 10;
```

The class A does not have an operation2() or an attribute2.

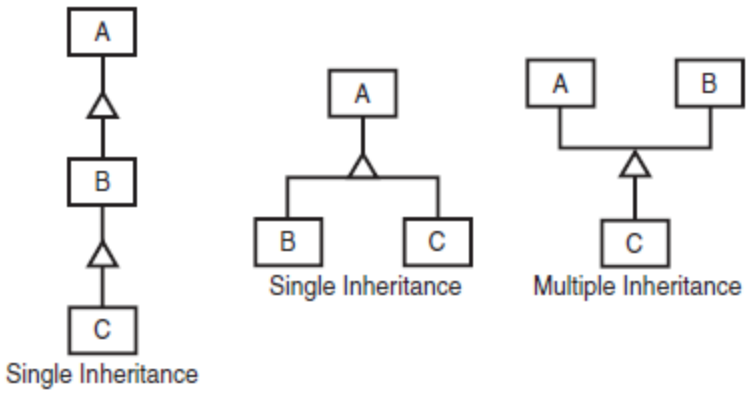


Figure 6.1 PHP does not support multiple inheritance.

6. Creating Classes, Attributes, and Operations in PHP

Structure of a Class
A minimal class definition looks like this:

```
class classname
{
//  properties and methods
Add();
}
```

To be useful, the classes need attributes and operations. You create attributes by declaring variables within a class definition using keywords that match their visibility: public, private, or protected. We will discuss this later in the documentation.

The following code creates class called classname with two public attributes, $attribute1 and $attribute2:

```
class classname{
        public $attribute1;
        public $attribute2;
}
```

You create operations by declaring functions within the class definition. The following code creates a class named classname with two operations that do nothing. The operation operation1() takes no parameters, and operation2() takes two parameters:

```
class classname{
        function operation1(){
        }
        function operation2($param1, $param2){
        }
}
```

7. Constructors

Constructor is nothing but a function defined in your php class. Constructor function automatically called when you will create object of the class. As soon as you will write $object = new yourClass() your constructor function of the class will be executed. In php4 you can create constructor by creating function with same name of your class. But from php5 you can also create constructor by defining magic function __construct. Please go through the below example of the constructor.

```
PHP 4 constructor(will work in php 5 also)
class interestCalculator
{
var $rate;
var $duration;
var $capital;
//Constructor of the class
function interestCalculator()
{
$this->rate = 3;
$this->duration = 4;
}
}
```

PHP5 constructor

```
class interestCalculator
{
public $rate;
public $duration;
public $capital;
//Constructor of the class
public function __construct()
```

```
{
$this->rate = 3;
$this->duration = 4;
}
}
```

Most classes have a special type of operation called a constructor. A constructor is called when an object is created, and it also normally performs useful initialization tasks such as setting attributes to sensible starting values or creating other objects needed by this object.

```
class classname
{
function __construct($param)
{
echo "Constructor called with parameter ".$param."<br />";
}
}
```

Although you can manually call the constructor, its main purpose is to be called automatically when an object is created.

So in short you cannot make your constructor private. If you will make your constructor private then you will receive an error.

8. Destructor

The opposite of a constructor is a destructor. They allow you to have some functionality that will be executed just before a class is destroyed, which will occur automatically when all references to a class have been unset or fallen out of scope.

Similar to the way constructors are named, the destructor for a class must be named __destruct(). **Destructors cannot take parameters.**

9. Instantiating Classes

After you have declared a class, you need to create an object—a particular individual that is a member of the class—to work with. This is also known as creating an instance of or instantiating a class. You create an object by using the new keyword.

```
class classname
{
function _construct($param)
{
echo "Constructor called with parameter ".$param."<br />";
}
}
$a = new classname("First");
$b = new classname("Second");
$c = new classname();
```

Because the constructor is called each time you create an object, this code produces the following output:

```
Constructor called with parameter First
Constructor called with parameter Second
Constructor called with parameter
```

10. Using Class Attributes

Within a class, you have access to a special pointer called $this. If an attribute of your **current class** is called $attribute, you refer to it as **$this->attribute** when either setting or accessing the variable from an operation within the class.

The following code demonstrates setting and accessing an attribute within a class:

```
class classname
{
public $attribute;
function operation($param)
{
$this->attribute = $param
echo $this->attribute;
}
}
```

Whether you can access an attribute from outside the class is determined by access modifiers, discussed later in this document. This example does not restrict access to the attributes, so you can access them from outside the class as follows:

```
class classname
{
public $attribute;
}
$a = new classname();
$a->attribute = "value";
echo $a->attribute;
```

It is not generally a good idea to directly access attributes from outside a class. One of the advantages of an object-oriented approach is that it encourages **Encapsulation**. You can enforce this with the use of __get and __set functions.

```
class classname
{
public $attribute;
function __get($name)
{
return $this->$name;
}
function __set ($name, $value)
{
$this->$name = $value;
}
}

$a = new classname();
echo $a->name = 'harsh';
```

11. Controlling Access with private and public

PHP uses access modifiers. They control the visibility of attributes and methods, and are placed in front of attribute and method declarations. PHP supports the following three different access modifiers:

1. The default option is public, meaning that if you do not specify an access modifier for an attribute or method, it will be public. Items that are public can be accessed from inside or outside the class.

2. **The private access modifier means that the marked item can be accessed only from inside the class.** You might use it on all attributes if you are not using __get() and __set().You may also choose to make some methods private, for example, if they are utility functions for use inside the class only. **Items that are private will not be inherited** (more on this issue later in this document).

3. The protected access modifier means that the marked item can be accessed only from inside the class. It also exists in any subclasses; again, we return to this issue when we discuss inheritance later in this documentation. For now, you can think of protected as being halfway in between private and public.

The following code shows the use of the public, private and protected access modifier:

```
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


/**
 * Define MyClass2
 */
class MyClass2 extends MyClass
{
   // We can redeclare the public and protected method, but not private
   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
```

12. Calling Class Operations

You can call class operations in much the same way that you call class attributes. Say you have the class

```
class classname
{
function operation1()
{
}
function operation2($param1, $param2)
{
}
}
```

and create an object of type classname called $a as follows:

```
$a = new classname();

$a->operation1();
$a->operation2(12, "test");
```

If the operations return something, you can capture that return data as follows:

```
$x = $a->operation1();
$y = $a->operation2(12, "test");
```

You then call operations the same way that you call other functions: by using their name and placing any parameters that they need in brackets. Because these operations belong to an object rather than normal functions.

13. Controlling Visibility Through Inheritance with private and protected

You can use the access modifiers private and protected to control what is inherited.
If an attribute or method is specified as private, it will not be inherited. If an attribute or method is specified as protected, it will not be visible outside the class (like a private element) but will be inherited.

```
class A{
        private function operation1(){
                echo "operation1 called";
        }

        protected function operation2(){
                echo "operation2 called";
        }

        public function operation3(){
                echo "operation3 called";
```

```
                }
        }

class B extends A{
        function __construct(){
                $this->operation1();
                $this->operation2();
                $this->operation3();
        }
}
$b = new B;
```

14. Overriding

Basic meaning of overriding in oop is same as real word meaning. In real word meaning of overriding phenomena of replacing the same parental behavior in child. This is same in case of method overriding in oop. In oop meaning of overriding is to replace parent class method in child class. Or in simple technical word method overriding mean changing behavior of the method. In oop overriding is process by which you can re-declare your parent class method in child class. So basic meaning of overriding in oop is to change behavior of your parent class method.

Normally method overriding required when your parent class have some method, but in your child class you want the same method with different behavior. By overriding of method you can complete change its behavior from parent class. To implement method overriding in oop we commonly create same method in child class.

For instance, say you have a class A:

```
class A{
    public $attribute = "default value";
        function operation(){
            echo "Something<br />";
            echo “The value of \$attribute is ". $this->attribute."<br />";
        }
}
```

If you want to alter the default value of $attribute and provide new functionality for operation(), you can create the following class B, which overrides $attribute and operation():

```
class B extends A{
        public $attribute = "different value";
        function operation(){
                echo "Something else<br />";
                echo "The value of \$attribute is ". $this->attribute."<br />";
        }
}
```

Declaring B does not affect the original definition of A. Now consider the following two lines of code:

```
$a = new A();
$a -> operation();
```

These lines create an object of type A and call its operation() function. This produces Something The value of $attribute is default value proving that creating B has not altered A. If you create an object of

type B, you will get different output.

This code

```
$b = new B();
$b -> operation();
```

produces
Something else
The value of $attribute is different value

15. Preventing Inheritance and Overriding with final

PHP uses the keyword final. When you use this keyword in front of a function declaration, that function cannot be overridden in any subclasses. For example, you can add it to class A in the previous example, as follows:

```
class A
{
public $attribute = "default value";
final function operation()
{
echo "Something<br />";
echo "The value of \$attribute is ". $this->attribute."<br />";
}
}
```

Using this approach prevents you from overriding operation() in class B. If you attempt to do so, you will get the following error:

Fatal error: Cannot override final method A::operation()
You can also use the final keyword to prevent a class from being subclassed at all. To prevent class A from being subclassed, you can add it as follows:

final class A
{...}
If you then try to inherit from A, you will get an error similar to
Fatal error: Class B may not inherit from final class (A)

16. Implementing Interfaces

Interface in php can be implemented like other oop lanugage. You can create interface in php using keyword interface. By implementation of interface in php class you are specifying set of the method which classes must implement.

You can create interface in php using interface keyword. Rest of the things are typically identical to classes. Following is very small example of interface in php.

```
interface vehicle{
    public function move($b);
}
```

```
class car implements vehicle{
   public function move($b){
          echo $b;
   }
}

$car = new car();
$car->move(5);
```

You can only define method in interface with public accessibility. If you will use other than public visibility in interface then it will throw error. Also while defining method in your interface do not use abstract keyword in your methods.

You can also extend interface like class. You can extend interface in php using extends keyword.

```
interface vehicle{
  public function move($b);
}

interface airways extends vehicle{
 public function moveback($a);
}


class car implements airways{
 public function move($b){
          echo $b;
 }

 public function moveback($a){
          echo $a;
 }
}

$car = new car();
$car->move(5);
```

You can also extend multiple interface in one interface in php.

```
interface vehicle{
  public function move($b);
}

interface train{
  public function moveforward($b);
}

interface airways extends vehicle,train{
 public function moveback($a);
}

class car implements airways{
```

```
 public function move($b){
        echo $b;
 }

 public function moveback($a){
        echo $a;
 }

  public function moveforward($a){
        echo $a;
 }
}

$car = new car();
$car->move(5);
```

You can also implement more than one interface in php class.

```
interface vehicle{
   public function move();
}

interface train{
   public function moveback();
}

class test implements vehicle, train{
   public function move(){
         echo 1;
   }

   public function moveback(){
         //your function body
   }
}

$test = new test();
$test->move();
```

17. Using Per-Class Constants

PHP allows for per-class constants. This constant can be used without your needing to instantiate the class, as in this example:

```
class Math {
public $name = "harsh";
const pi = 3.14159;
}
echo " Math::pi = ".Math::pi."\n";
```

Output:
Math::pi = 3.14159

You can access the per-class constant by using the :: operator to specify the class the constant belongs

to, as done in this example.

18. Implementing Static Methods

PHP allows the use of the static keyword. It is applied to methods to allow them to be called without instantiating the class.
This is the method equivalent of the per-class constant idea. For example, consider the Math class created in the preceding section. You could add a squared() function to it and invoke it without instantiating the class as follows:

```
class Math
{
static function squared($input)
{
return $input*$input;
}
}
echo Math::squared(8);
```

Output:
64

Note that you cannot use the this keyword inside a static method because there may be no object instance to refer to.

19. Checking Class Type and Type Hinting

The **instanceof keyword** allows you to check the type of an object. You can check whether an object is an instance of a particular class, whether it inherits from a class, or whether it implements an interface.

instanceof keyword Example:-

```
class A{
  // class contains properties and methods.
}
$obj = new A();

if ($obj instanceof A) {
  echo 'A';
}
```

This code will output: - A

Example of Type Hinting:-

Normally, when you pass a parameter to a function in PHP, you do not pass the type of that parameter. With **class type hinting**, you can specify the type of class that ought to be passed in, and if that is not the type actually passed in, an error will be triggered. The type checking is equivalent to instanceof. For example, consider the following function:

```
function check_hint(B $someclass)
```

```
{
//...
}
```

This example suggests that $someclass needs to be an instance of class B. If you then pass in an instance of class A as check_hint($a);

you will get the following fatal error:
Fatal error: Argument 1 must be an instance of B
Note that if you had hinted A and passed in an instance of B, no error would have occurred because B inherits from A.

Another Example:-

```
class MyClass
{
   /**
   * A test function
   *
   * First parameter must be an object of type OtherClass
   */
   public function test(OtherClass $otherclass) {
     echo $otherclass->var;
   }


   /**
   * Another test function
   *
   * First parameter must be an array
   */
   public function test_array(array $input_array) {
     print_r($input_array);
   }
}

// Another example class
class OtherClass {
   public $var = 'Hi how are you?';
}
$obj = new OtherClass();
$objtwo = new MyClass();
$objtwo->test($obj);
```

Output:
Hi how are you?

20. Late Static Bindings

Introduced in PHP 5.3, late static bindings allow references to the called class within the context of a static inheritance; parent classes can use static methods overridden by child classes. The following basic example from the PHP Manual shows a late static binding in action:

Example #1 self:: usage

```
class A {
   public static function who() {
      echo __CLASS__;
   }
   public static function test() {
      self::who();
   }
}

class B extends A {
   public static function who() {
      echo __CLASS__;
   }
}

B::test();
Output:
A
```

Example #2 static:: simple usage

```
class A {
        public static function who() {
                echo __CLASS__;
        }
        public static function test() {
                static::who(); // Here comes Late Static Bindings
        }
}
class B extends A {
        public static function who() {
                echo __CLASS__;
        }
}
B::test();
```

Output:
B

Allowing references to classes called at runtime, regardless if they have been overridden, brings additional object functionality to your classes.

21. Cloning Objects

The clone keyword, which allows you to copy an existing object, can also be used in PHP. For example,
$c = clone $b;
Creates a copy of object $b of the same class, with the same attribute values.
You can also change this behavior. If you need nondefault behavior from clone, you need to create a method in the base class called __clone().This method is similar to a constructor or destructor in that you do not call it directly. It is invoked when the clone keyword is used as shown here. Within the __clone() method, you can then define exactly the copying behavior that you want.

The nice thing about __clone() is that it will be called after an exact copy has been made using the default behavior, so at that stage you are able to change only the things you want to change.

The most common functionality to add to __clone() is code to ensure that attributes of the class that are handled as references are copied correctly. If you set out to clone a class that contains a reference to an object, you are probably expecting a second copy of that object rather than a second reference to the same one, so it would make sense to add this to __clone().

You may also choose to change nothing but perform some other action, such as updating an underlying database record relating to the class.

Example:

```
class SubObject
{
   static $instances = 0;
   public $instance;

   public function __construct() {
      $this->instance = ++self::$instances;
   }

   public function __clone() {
      $this->instance = ++self::$instances;
   }
}

class MyCloneable
{
   public $object1;
   public $object2;

   function __clone()
   {
      // Force a copy of this->object, otherwise
      // it will point to same object.
      $this->object1 = clone $this->object1;
   }
}

$obj = new MyCloneable();

$obj->object1 = new SubObject();
$obj->object2 = new SubObject();
```

```
$obj2 = clone $obj;


print("Original Object:\n");
print_r($obj);

print("Cloned Object:\n");
print_r($obj2);
```

The above example will output:

```
Original Object:
    MyCloneable Object
    (
       [object1] => SubObject Object
          (
             [instance] => 1
          )

       [object2] => SubObject Object
          (
             [instance] => 2
          )

    )
    Cloned Object:
    MyCloneable Object
    (
       [object1] => SubObject Object
          (
             [instance] => 3
          )

       [object2] => SubObject Object
          (
             [instance] => 2
          )

    )
```

22. Using Abstract Classes

What is abstract Classes
As from name it seem like something that is hidden. Yes nature of the abstract classes are same. Abstract classes are those classes which can not be directly initialized. Or in other word we can say that you can not create object of abstract classes.

Abstract classes always created for inheritance purpose. You can only inherit abstract class in your child class.

Abstract classes in php are similar like other oop languages. You can create abstract classes in php using abstract keyword. Once you will make any class abstract in php you can not create object of that class.

```
abstract class car{

   public function move(){
          echo 1;
   }
}
$car = new car();//this will throw error in php
```

Abstract classes in php are only for inheriting in other class.

```
abstract class vehicle{

   public function move(){
          echo 1;
   }
}

class car extends vehicle{

}

$car = new car();
$car->move();
```

Abstract method

```
abstract class vehicle{
   abstract protected function move($a , $b);
}

class car extends vehicle{
   protected function move($x,$y){
          echo $x+$y;
   }

   function testing(){
     echo $this->move(1,2);
   }
}
$car = new car();
$car->testing();


abstract class vehicle{

   abstract protected function move();
   abstract public function moveback();
   // abstract private function moveforward(); //this will trhow error
}
```

```
class car extends vehicle{

   public function move(){
         //body of your function
   }

   public function moveback(){
         //body of your function
   }

   protected function moveforward(){
         //body of your function
   }
}
$car = new car();
```

// Private declaration of the abstract method will always throw error. Because private method is available only in the same class context.

23. Overloading Methods with __call()

Overloading in oop is same as overloading in real word. In real word overloading means assigning extra work to same machine or person. In oop method overloading is same. By process of method overloading you are asking your method to some extra work. Or in some cases we can say some different work also.

Normally method overloading in oop is managed on the basis of the argument passed in function. We can achieve overloading in oop by providing different argument in same function.

We previously looked at a number of class methods with special meanings whose names begin with a double underscore (__), such as __get(), __set(), __construct(), and __destruct().

Another example is the method __call(), which is used in PHP to implement method overloading.

Method overloading is common in many object-oriented languages but is not as useful in PHP because you tend to use flexible types and the (easy-to-implement) optional function parameters instead.

As we know that we cannot implement overloading by create 2 functions in with same name in class. So to implement overloading in php we will take help of magic method __call. Magic method __call invoked when method called by class object is not available in class. So here we will not create method exactly and will take help of __call method. Now call method will provide us 2 argument, 1st name of the method called and parameter of the function. Now with the help of either switch, case or if else we will implement overloading in php. Following is very simple example of overloading in php.

To use it, you implement a __call() method, as in this example:

```
class overload{
public function __call($method, $p){
                if ($method == "display") {
                        if (is_object($p[0])) {
                                $this->displayObject($p[0]);
                        } else if (is_array($p[0])){
                                print_r($p);
```

```
                        } else {
                        $this->displayScalar($p[0]);
                        }
                }
        }
}
$ov = new overload;
echo $ov->display(array(1, 2, 3));
echo $ov->display('cat');
```

The __call() method should take two parameters .The first contains the name of the method being invoked, and the second contains an array of the parameters passed to that method. You can then decide for yourself which underlying method to call. In this case, if an object is passed to method display(), you call the underlying displayObject() method; if an array is passed, you call displayArray(); and if something else is passed, you call displayScalar().

To invoke this code, you would first instantiate the class containing this __call() method (name it overload) and then invoke the display() method, as in this example:

```
$ov = new overload;
$ov->display(array(1, 2, 3));
$ov->display('cat');
```

The first call to display() invokes displayArray(), and the second invokes displayScalar().

24. Using __autoload()

Another of the special functions in PHP is __autoload(). It is not a class method but a standalone function; that is, you declare it outside any class declaration. If you implement it, it will be automatically called when you attempt to instantiate a class that has not been declared.

The main use of __autoload() is to try to include or require any files needed to instantiate the required class. Consider this example:

```
function __autoload($name)
{
include_once $name.".php";
}
```

This implementation tries to include a file with the same name as the class.

25. Implementing Iterators and Iteration (code)

26. Converting Your Classes to Strings

If you implement a function called __toString() in your class, it will be called when you try to print the class, as in this example:

```
$p = new Printable;
echo $p;
```

Whatever the __toString() function returns will be printed by echo. You might, for instance, implement it as follows:

```
class Printable
{
public $testone;
public $testtwo;
public function __toString()
{
return(var_export($this, TRUE));
}
}
```

(The var_export() function prints out all the attribute values in the class.)

Another Example:-

```
class Printable
{
public $testone;
public $testtwo;
public function __toString()
{
return(var_export($this, TRUE));
}
}

$p = new Printable;

$str = (string) $p;


if (is_string($str)) {
   echo "\$str is string\n";
} else {
   echo "is not a string\n";
}

$arr = (array) $p;

if (is_array($arr)) {
   echo "\$arr is Array\n";
} else {
   echo "is not a Array\n";
}
```

This code produces:
$str is string $arr is Array

27. Using the Reflection API

PHP's object-oriented features also include the reflection API. Reflection is the ability to interrogate existing classes and objects to find out about their structure and contents. This capability can be useful when you are interfacing to unknown or undocumented classes, such as when interfacing with encoded PHP scripts.

```
require_once("page.inc");

$class = new ReflectionClass("Page");
echo "<pre>".$class."</pre>";
```

28. Classes/Object Functions

Table of Contents

1) __autoload — Attempt to load undefined class
2) call_user_method_array — Call a user method given with an array of parameters [deprecated]
3) call_user_method — Call a user method on an specific object [deprecated]
4) class_alias — Creates an alias for a class
5) class_exists — Checks if the class has been defined
6) get_called_class — the "Late Static Binding" class name
7) get_class_methods — Gets the class methods' names
8) get_class_vars — Get the default properties of the class
9) get_class — Returns the name of the class of an object
10) get_declared_classes — Returns an array with the name of the defined classes
11) get_declared_interfaces — Returns an array of all declared interfaces
12) get_declared_traits — Returns an array of all declared traits
13) get_object_vars — Gets the properties of the given object
14) get_parent_class — Retrieves the parent class name for object or class
15) interface_exists — Checks if the interface has been defined
16) is_a — Checks if the object is of this class or has this class as one of its parents
17) is_subclass_of — Checks if the object has this class as one of its parents
18) method_exists — Checks if the class method exists
19) property_exists — Checks if the object or class has a property
20) trait_exists — Checks if the trait exists