Object Oriented Properties

- **Inheritance-** A PHP class can acquire the properties of another class through inheritance. A PHP inheritance has the following generic structure-

```
<?php
class Person{
    var $name="ABC DEF";
    var $age=20;
    function display()
    {
      echo "$this->name"."  is "."$this->age"." years old<br/>";
    }
}

class Student extends Person{
    var $clas=12;
    var $roll=1;
    function display()
    {
     echo "reads in class "."$this->clas"." and roll no ". "$this->roll";
    }
}

  $obj1=new Person();
  $obj2=new student();
  $obj1->display();
  $obj2->display();

?>
```

- **Access Modifiers-** In PHP, we have three access specifiers or modifiers. They are-
  1. **Public-** members are accessible everywhere.
  2. **Private-** members are accessible only within this class.
  3. **Protected-** members are accessible within this class and its subclass..

- Consider the following example to understand the concept clearly-

```
class InheritanceDemo{
        public $x=10;
        private $y=20;
        protected $z=30;
```

```
    function display()    // note that the function is public by default
    {
       echo  $this->x;
       echo  $this->y;
       echo  $this->z;
    }
}   //end of class

$obj=new InheritanceDemo();
echo "The value of public x is {$obj->x}<br/>";

 /* you will get fatal error if you wish to execute the following instructions as they
    cannot be executed outside the class.
echo "The value of public y is {$obj->y}<br/>";   error
echo "The value of public z is {$obj->z}<br/>";   error
*/
$obj->display();    //no problem since it is public member
?>
```

- **Static modifiers-** They are similar to C++/JAVA class variables. That is, they are declared once for the class and accessed without declaring the instance.
  - Consider the following code fragment to understand the static modifier-

```
<?php
  class StaticDemo{
  static $count_people=1;

  static function display()
  {
    self::$count_people++;
    echo "You are within a static method and the value of $count-people is ";
    echo  self::$count_people;

  }
}// end of class

echo StaticDemo::$count_people."<br/>";  // class name scope resolution attribute name

StaticDemo::display();  // class name scope resolution function name
?>
```

  - Value of static variable persists (retains) during method calls.
  - **Note:**
    1. With static methods we can't use $this reference.
    2. We may get an error like "Paamayim Nekudotayim" (which means double colon in Hebrew) whenever we use a scope resolution operator in a wrong way.

3. In order to access a static member of a super class from a sub class, you can use the 'parent' keyword in the place of 'self' keyword.

- **Constructor & Destructor-**
  - In PHP 4 Constructor has the name as the class name.
  - In PHP 5 a constructor is declared as _ _ construct() .
  - Consider the following program-

```
<?php
class Myclass{
    public $x;
  function _ _construct($y)      //constructors
  {
    $this->x=$y;
  }

  function display()
  {
   echo "$this->x";
  }
}

  $obj=new Myclass(10);     //constructor
  $obj->display();              //output is 10
?>
```

  - A subclass can use a parent keyword to invoke the super class constructor as follows-
    parent::_ _construt(); // within the sub class constructor
  - A destructor, hardly used in PHP, is declared as follows-

```
function _ _destruct(){
   // some code
}
```

- **Cloning (copying) an object-**

```
<?php
  class clone_demo{
  public $x=1;
  function __construct()
   {
     echo "Object is created";
   }
  function __clone()            // this method is executed while cloning
   {
     echo "<br/>Object is copied";
   }
```

```
}
$obj1=new clone_demo();
$obj2=$obj1;   //creating reference

$obj2->x=10;   //udpating reference

echo "Value of the refernce object is ".$obj1->x."<br/>"; // the original object is changed


$obj3=clone $obj1;
$obj3->x=20;

echo "The value of clone object is ".$obj3->x."<br/>";
echo "The value in original object is ".$obj1->x."<br/>";

?>
```

- **Comparing objects through = = and = = = operator-**

Objects	Operator = =	Operator = = =
References	true	true
Instances with matching attributes	true	false
Instances with different attributes	false	false