# PROGRAMACIÓN ORIENTADA A OBJETOS EN PHP

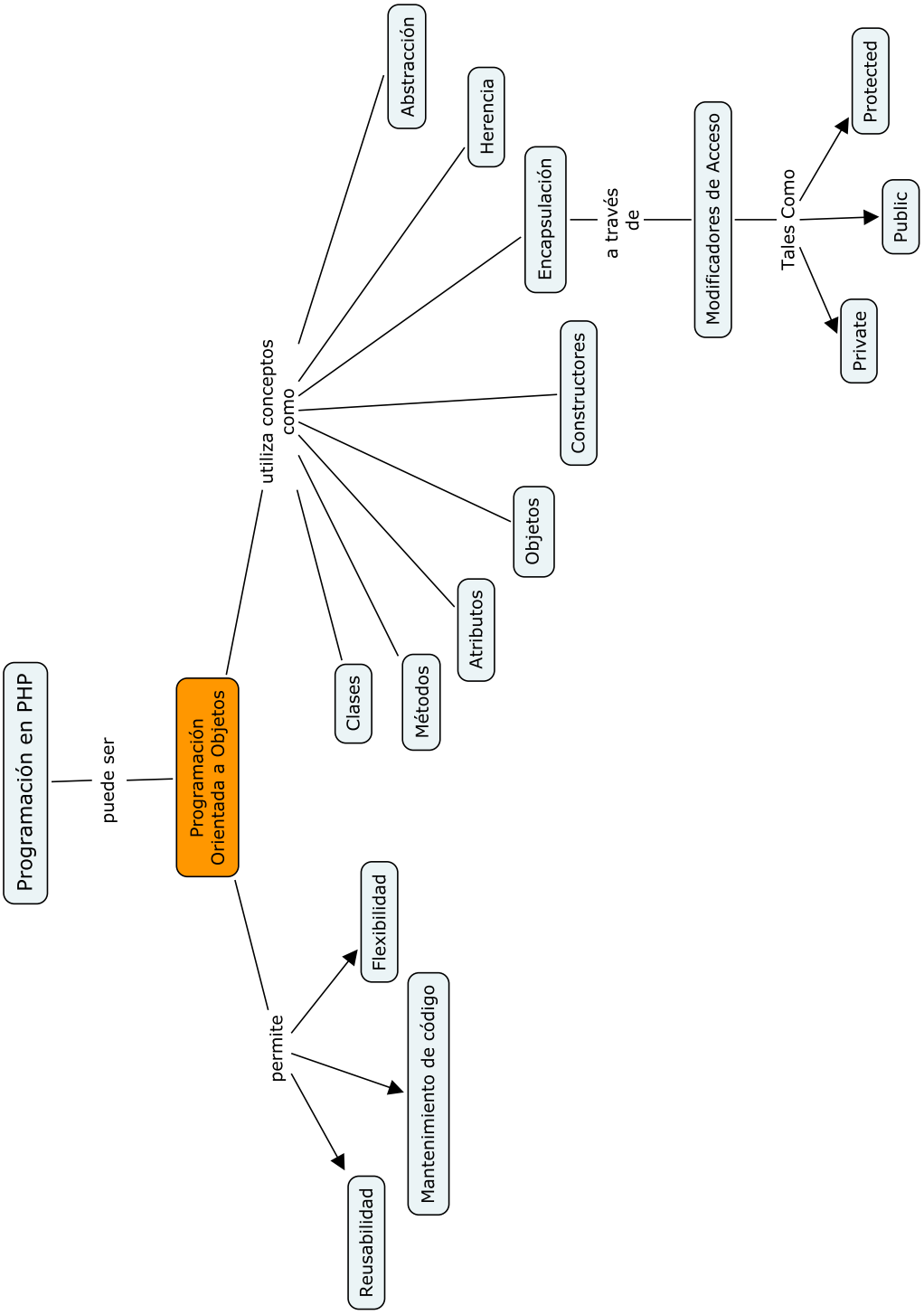
## Programa: ANÁLISIS Y DESARROLLO DE SISTEMAS DE INFORMACIÓN

### Duración estimada de estudio (horas): 6 horas

### Estructura de contenidos

## Mapa Conceptual

### Programación Orientada a Objetos en PHP.




## Introducción

Al revisar los conceptos relacionados con la construcción de sistemas de información es relevante explorar, reconocer y desarrollar competencias en el uso de la Programación Orientada a Objetos (POO). La aplicación de recursos y elementos como: Clase, Objeto, Herencia, Polimorfismo, Abstracción y Encapsulación son parte esencial de la POO.

El presente objeto de aprendizaje, presenta los conceptos y recursos que le permitan comprender los fundamentos de la Programación Orientada a Objetos utilizando el lenguaje de desarrollo PHP.

# 1. Definición de una clase en PHP

Una clase inicia con la palabra reservada class en minúscula y después el nombre.

```
class NombreClase
{

}
```

## Declaración de Clases Abstractas

Las clases abstractas son aquellas que no necesitan ser instanciadas, pero si pueden ser heredadas.  Se definen anteponiendo la palabra abstract.

```
<¿php

abstract class NombreClase{

//No necesita ser instanciada

}

?>
```

## Declaración de Clases Final

PHP incorpora clases finales que no pueden ser heredadas por otra. Se definen anteponiendo la palabra reservada final.

```
<¿php
 final class NombreClase{
  //esta clase no puede ser heredada
 }
?>
```

## 1.1 Estructura General de una Clase en PHP.

La siguiente figura presenta la estructura general de una clase, la definición de los atributos, seguido del constructor, como elemento opcional el destructor y por último la definición de los métodos de la clase.

```
1   <¿php
2
3   class NombreClase
4    {
5
6     //Definición de Atributos
7
8     //Definición Constructor
9
10    //Definición de destructor (opcional)
11
12    //Definición de Métodos
13
14   }
15  ?>
16
```

Ahora vamos a mostrar la implementación de una clase denominada Persona como ejemplo.

```php
<¿php
class Persona
{
    //Definición de Atributos
    private $nombre;

    //Constructor
    public function Persona($nombre)
    {
        $this->nombre=$nombre;
    }

   //Definición de Métodos
    public function getNombre()
    {
        return $this->nombre;
    }
    public function setNombre($value)
    {
        $this->nombre=$value;
    }

    public function leer($libro)
    {
        //aquí va el código del método
    }
 }
?>
```

## 1.2 Los Atributos

Los atributos son las características, cualidades, propiedades distintivas de cada clase. Contienen información sobre el objeto. Determinan la apariencia, estado y demás particularidades de la clase. Varios objetos de una misma clase tendrán los mismos atributos pero con valores diferentes.

Cuando creamos un objeto de una clase determinada, los atributos declarados por la clase son localizadas en memoria y pueden ser modificados mediante los métodos.

Lo más conveniente es que los atributos sean privados para que solo los métodos de la clase puedan modificarlos.

**Definición:**

```
//Definición de Atributos
 private $identificación;
 private $nombre;
 private $fechanacimiento;
```

## 1.3 Los constructores

El constructor es un método especial de una clase. El objetivo fundamental del constructor es inicializar los atributos del objeto que creamos.

Se pueden definir varios constructores (sobrecarga de constructores), que permitan crear el objeto de diferentes maneras.

**Definición:**

Se pueden definir de dos formas:

1. Mediante un método llamado **__construct**

```
//Constructor
 public function __construct($nombre)
 {
    $this->nombre=$nombre;
 }
```

2. Mediante un método llamado igual que el nombre de la clase.

```
//Constructor
 public function Persona($nombre)
 {
    $this->nombre=$nombre;
 }
```

**Otras características de los constructores son:**

- El constructor se ejecuta inmediatamente luego de crear un objeto y no puede ser llamado nuevamente.
- Un constructor no puede retornar dato.
- Un constructor puede recibir parámetros que se utilizan normalmente para inicializar atributos.
- El constructor es un método opcional, de todos modos es muy común definirlo.

## 1.4 Los Métodos

Los métodos son como las funciones en los lenguajes estructurados, pero están definidos dentro de una clase y operan sobre los atributos de dicha clase.

Los métodos de una Clase son los que nos permiten definir las funcionalidades o responsabilidades de la clase. Para ello debemos preguntarnos que puede hacer la clase.

El objetivo de un método es ejecutar las actividades que tiene encomendada la clase a la cual pertenece.

Los atributos de un objeto se modifican mediante llamadas a sus métodos.

**Ejemplo Definición de Métodos**

```
//Definición de Métodos
 public function getNombre()
 {
    return $this->nombre;
 }

 public function setNombre($value)
 {
    $this->nombre=$value;
 }
```

En los ejemplos anteriores el primer método retorna un valor para ello se utiliza la sentencia:

```
return $variable;
```

El segundo Método recibe un parámetro y este parámetro es asignado al atributo nombre.

```
$this->nombre=$value;
```

## 1.5 Encapsulación

La encapsulación en un programa se da cuando se definen los atributos de una clase con el nivel de acceso más restrictivo, para que el acceso o la modificación de los atributos de esa clase sólo sean posibles a través de sus métodos

```php
<¿php
class Persona
{
    //Definición de Atributos
    private $identificación;
    private $nombre;
    private $fechanacimiento;

    //Constructor
    public function Persona($nombre)
    {
        $this->nombre=$nombre;
    }

  //Definición de Métodos
    public function getNombre()
    {
        return $this->nombre;
    }
    public function setNombre($value)
    {
        $this->nombre=$value;
    }

    public function getIdentificación()
    {
        return $this->identificación;
    }
    public function setidentificación($value)
    {
        $this->nombre=$value;
    }

    public function getFechaNacimiento()
    {
        return $this->fechaNacimiento;
    }
    public function setFechaNacimiento($value)
    {
        $this->fechaNacimiento=$value;
    }
}
?>
```

Encapsulación

Métodos

## 1.6 Modificadores de Acceso

**Public:** Cuándo un atributo o un método es definido como public se puede acceder a él directamente desde donde se haya instanciado el objeto.

**Ejemplos:**

```
public $atributo;

public function metodo()
  {
     return $valor;
  }
```

**Private:** Cuando un atributo o un método es definido como private, solo se puede acceder a ellos desde la misma clase.

**Ejemplos:**

```
//Definición de Atributos
private $identificación;
private $nombre;
private $fechanacimiento;

//Constructor
private function operacion($valor1)
{
    return $valor*2;
}
```

**Protected:** Cuando un atributo o método es definido como protected, solo se puede acceder a él sin ninguna restricción desde la clase o desde sus heredados.

**Ejemplos:**

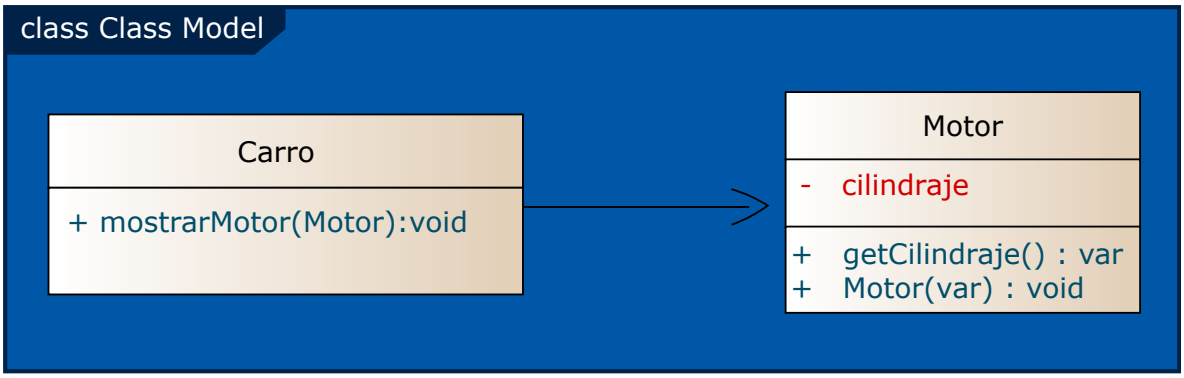
```
//Definición de Atributos
protected $identificación;
protected $nombre;
```

# 2. Representación de las Relaciones entre Clases

## 2.1 Relación de Dependencia

Es una relación de uso entre dos entidades, una de las clases usa a la otra.

La relación de dependencia es cuando una clase depende de la funcionalidad que ofrece otra clase

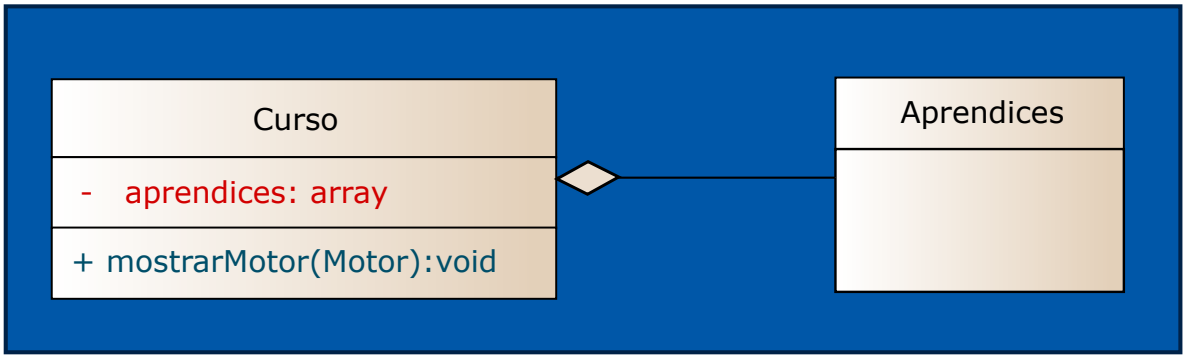



**Ahora como se traduce en código PHP:**

```php
<¿php
require_once ('Motor.php');

class Carro
{

   function mostrarMotor(Motor $objetoMotor)
   {
      echo $objetoMotor->getCilindraje();
   }

}
?>

```

## 2.2 Relación de Agregación

Es una relación de asociación donde una de las clases forma parte del todo correspondiente a la otra clase.



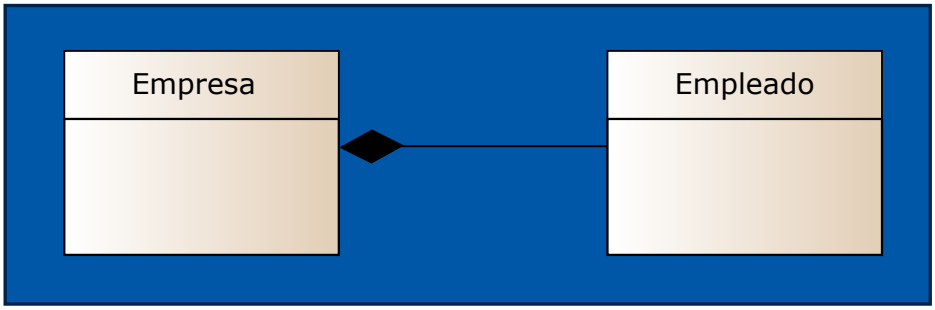

**Ahora como se traduce en PHP:**

```php
<?php
require_once ('Aprendices.php');
class Curso
{
    //definiendo un arreglo donde se van a guardar los aprendices
    private $aprendices = array();

    function agregarAprendices(Aprendices $aprendiz)
    {
        //se agrega al arreglo $aprendices el objeto aprendiz
        $this->aprendices[] = $aprendiz;
    }
}

$objCurso = new Curso();
$objCurso->agregarAprendices(new Aprendices());
$objCurso->agregarAprendices(new Aprendices());
$objCurso->agregarAprendices(new Aprendices());
$objCurso->agregarAprendices(new Aprendices());
$objCurso->agregarAprendices(new Aprendices());
/*Como ejemplo en el curso anterior solo se agregaron
5 aprendices */
?>
```

### 2.3 Relación de Composición

Similar a la relación de agregación solo que en está relación existe una relación de vida.

**Por ejemplo:**




En esta relación la empresa existe si tiene empleados. El código en PHP no cambia en nada frente a la relación de agregación.

## 3 Los objetos

Para crear un objeto, es necesario emplear alguno de los constructores de la clase. Cuando se esta creando un objeto se esta instanciando la clase.

Desde un archivo php podemos tener el siguiente código para crear un objeto:

```
$objeto = new Clase();

$objeto = new Clase($parametros);
```

En la creación del objeto encontramos la sentencia new y después va el nombre de la clase. Recordemos que al momento de instanciar se llama automáticamente a los constructores que tenga la clase. Por lo tanto podemos crear un objeto de diferentes formas, dependiendo del número de constructores.

## 3.1 Ejemplo de construcción de objetos

El ejemplo planteado permite crear una página php que instancia dos objetos de la clase persona y muestra el nombre de cada uno de ellos.

**En la siguiente imagen se relaciona el ejemplo completo:**

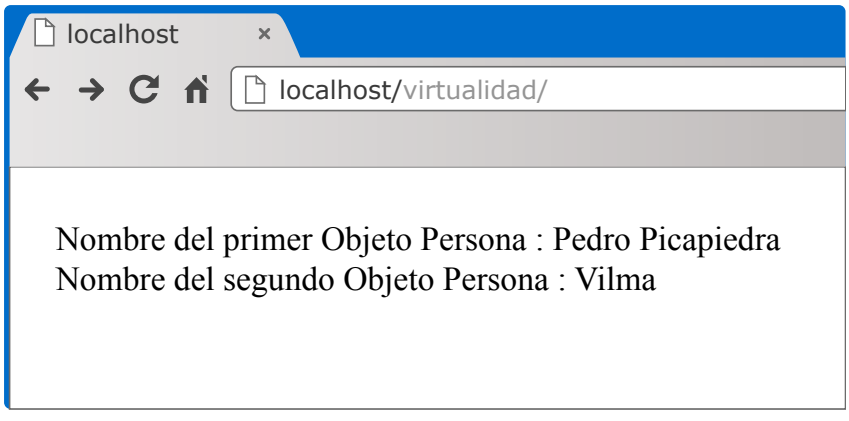
```php
<¿php
//incluimos el archivo donde esta el código de la clase Persona
   include "Persona.php";

//Instanciamos el primer Objeto
   $objPersona=new Persona("Pedro Picapiedra");//instanciamos

/* Para poder mostrar en pantalla el nombre de la persona debemos utilizar el
método gerNombre(), ya que el atributo $nombre es privado.*/


//Ahora Instanciamos otra vez la clase persona
   $objPersona=new Persona("Vilma");
   echo "<br> Nombre del segundo obeto Persona : ". $objPersona->getNombre();
?>
```

**Explicación línea por línea.**

| | |
|---|---|
| `<?php` | Código de inicio del código php. |
| `Include "Persona.php"` | La forma como incluimos un archivo php, aquí estamos incluyendo el archivo Persona.php que contiene el código de la clase persona. |
| `$objPersona = new Persona("Pedro Picapiedra")` | Esta línea crea un objeto llamado $objPersona y estamos pasando como parámetro el valor de Pedro Picapiedra. Veamos la relación con la clase<br>`//Constructor`<br>`public function Persona($nombre)`<br>`{`<br>`  $this->nombre=$nombre;`<br>`}`<br>`$objPersona=new Persona("Pedro Picapiedra"); //instanciamos` |

| | |
|---|---|
| `echo "<br> Nombre del Primer obeto Persona : ".$objPersona->getNombre();` | Esta línea obtiene el nombre de la persona mediante el método getNombre() de la clase persona y lo imprime en pantalla. Se debe utilizar ese método debido a que el atributo nombre está definido como private.<br>Veamos la relación<br>`//Definición de Métodos`<br>`public function getNombre()`<br>`{`<br>`  return $this->nombre;`<br>`}`<br>`echo "<br> Nombre del Primer objeto Persona :" .$objPersona->getNombre();` |
| ¿> | Etiqueta de cierre del código php |

**El resultado final al ejecutar el código es el siguiente:**




## 3.2 Herencia:

La herencia es la que me permite crear nuevas clases partiendo de clases ya existentes. En la Herencia existe el concepto de SuperClase o Clase padre y la clase que hereda se le conoce como clase Hija.

Definición de una clase que hereda de otra

**Ejemplo:**

```
class Hija extends Padre
{

}
```

Como ejercicio vamos a crear una clase Estudiante que hereda de la Clase persona.

```
//Definición de Atributos
protected $nombre;
```

Para ello primero vamos a modificar la clase Persona y vamos a declarar el atributo nombre como protected.

```
1  <¿php
2  include "Persona.php"; //la incluimos para poder acceder a ella
3  class Estudiante extends Persona
4  {
5     //Atributo curso: Que curso estudia el Estudiante.
6     private $curso;
7
8     //Método que obtiene el curso que estudia el estudiante.
9     public function getCurso()
10    {
11      return $this->curso;
12    }
13
14    //Método que permite cambiar el curso del estudiante.
15    public function setCurso($value)
16    {
17      $this->curso=$value;
18    }
19
20    //Constructor Clase Estudiante
21    public function Estudiante($nombre, $curso)
22    {
23    /* como iniciamos el atributo nombre que lo hereda de la
24    clase Persona para ello el atributo nombre debe ser declarado
25    como protected.*/
26      $this->nombre=$nombre;
27      $this->curso=$curso;
28    }
29 }
30 ?>
31
```

**Ahora vamos a crear un archivo donde podamos crear un objeto estudiante.**

```
<?php
//incluimos el archivo donde está el código de la clase Persona
include "Estudiante.php";
//Instanciamos para crear el primer objeto tipo Estudiante

$objEstudiante=new Estudiante(
```

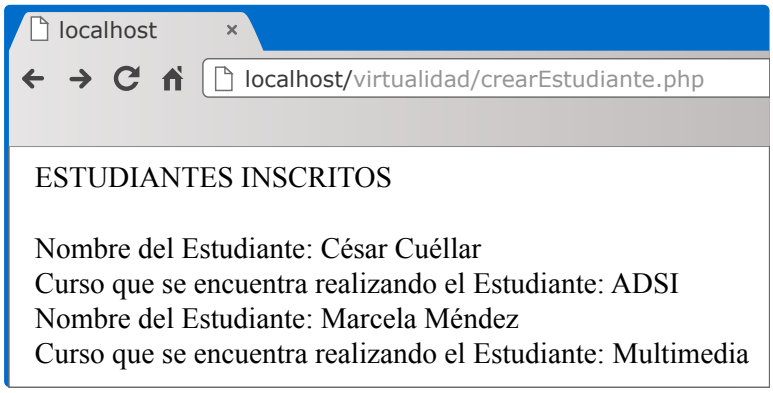
Estudiante(**$nombre**, $curso)

En la imagen anterior podemos ver en el momento de codificar los parámetros necesarios solicitados por el constructor de la clase Estudiante. Por lo anterior debemos de ingresar el nombre del Estudiante y el Curso.

**Código completo de la Implementación.**

```
1  <html>
2  <meta http-equiv="Contect-Type" contect="text/html; charset=utf-8">
3  <?php
4
5  //incluimos el archivo donde está el código de la clase Persona
6  include "Estudiante.php";
7  //Instanciamos para crear el primer objeto tipo Estudiante
8
9  $objEstudiante=new Estudiante("César Cuéllar", "DASI");
10
11 //creamos otro objeto de tipo Estudiante
12 $objEstudiante=new Estudiante("Marcela Médez", "Multimedia");
13
14 echo "ESTUDIANTES INSCRITOS<BR>";
15
16 /*Vamos ahora a imprimir en pantalla los Datos de los Estudiantes
17 accedemos al método getNombre() que lo hereda de la clase persona*/
18
19 echo "<br> Nombre del Estudiante : ". $objEstudiante->getNombre();
20
21 /*Ahora vamos a imprimir en pantalla el Curso, para ello
22 accedemos al método getCurso() propio de la clase Estudiante.*/
23 echo "<br> Curso que se encuentra realizando el Estudiante : ". $objEstudiante->getNombre();
24
25 echo "<br> Nombre del Estudiante : ". $objEstudiante2->getNombre();
26 echo "<br> Curso que se encuentra realizando el Estudiante : ". $objEstudiante2->getNombre();
27
28 ?>
```

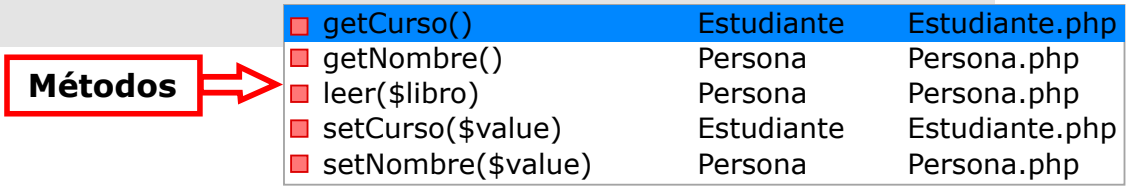
### Resultado de la Implementación




## 3.3 Ejecución de Métodos

Después de que se ha creado un objeto se puede acceder a los métodos públicos de la clase.

```
$objEstudiante=new Estudiante("César Cuéllar", "DASI");

$objEstudiante->
```




Nos muestra los métodos públicos propios de la clase o nos muestra los métodos heredados de alguna clase que sean declarados como protected.

# 4 Ejercicio de ejemplo completo

## Enunciado

Se requiere hacer una aplicación que permita calcular el salario a pagar a los empleados de una empresa. En la empresa hay dos tipos de empleados unos contratistas y otros de planta.

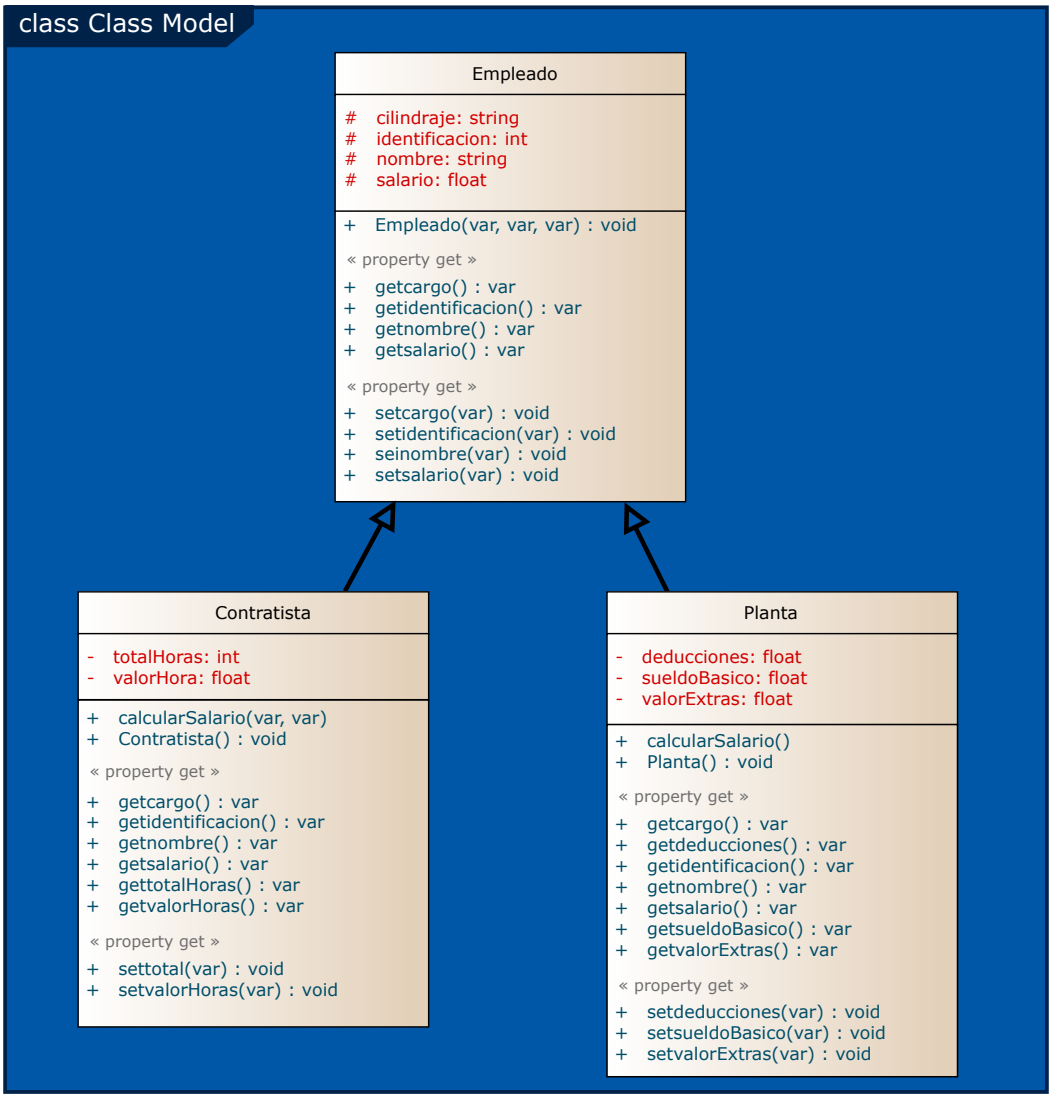
A los contratitas el salario depende del valor de la hora.

**Formula:** Salario = ValorHora * TotalHoras

El salario de los de planta depende del sueldo básico mensual.

**Formula** = SalarioBasico + HorasExtras – deducciones.

## Diagrama de Clases



## Código Clase Empleado

```php
<¿php
/**
 * @author César Cuéllar
 * @version 1.0
 * @created 20-oct-2013 07:47:27 a.m.
 */
class Empleado
{
protected $cargo;
protected $identificacion;
protected $nombre;
protected $salario;

//Constructor
/**
 * @param cargo, @param salario, @param nombre, @param
identificacion
 */
public function Empleado($identificacion,$nombre,$cargo)
{
$this->identificacion=$identificacion;
$this->nombre=$nombre;
$this->cargo=$cargo;
}

protected function getCargo()
{
return $this->cargo;
}

protected function getIdentificacion()
{
return $this->identificacion;
}

protected function getNombre()
{
return $this->nombre;
}

```

```php

protected function getSalario()
{
return $this->salario;
}

/** @param newVal */
protected function setCargo($newVal)
{
$this->cargo = $newVal;
}

/** @param newVal */
protected function setIdentificacion($newVal)
{
$this->identificacion = $newVal;
}

/** @param newVal */
protected function setNombre($newVal)
{
$this->nombre = $newVal;
}

/** @param newVal */
protected function setSalario($newVal)
{
$this->salario = $newVal;
}
}
?>

```

## Código Clase Contratista

```php
<¿php
/**
 * @author César Cuéllar
 * @version 1.0
 * @created 20-oct-2013 07:47:29 a.m.
 */
class Contratista extends Empleado
{
private $totalHoras;
private $valorHora;

//Constructor
public function
Contratista($identificacion,$nombre,$cargo)
{
//ejecutamos el connstructor de la clase Padre Empleado
parent::__construct($identificacion,$nombre,$cargo);
}

/** @param totalHoras, @param valorHora */
public function calcularSalario($valorHora, $totalHoras)
{
$this->salario= $valorHora*$totalHoras;
}

public function getTotalHoras()
{
return $this->totalHoras;
}

public function getValorHora()
{
return $this->valorHora;
}

public function getCargo()
{
return $this->cargo;
}

```

```
40
41  public function getIdentificacion()
42  {
43  return $this->identificacion;
44  }
45  public function getNombre()
46  {
47  return $this->nombre;
48  }
49
50  public function getSalario()
51  {
52  return $this->salario;
53  }
54
55  /** @param newVal */
56  public function setTotalHoras($newVal)
57  {
58  $this->totalHoras = $newVal;
59  }
60
61  /** @param newVal */
62  public function setValorHora($newVal)
63  {
64  $this->valorHora = $newVal;
65  }
66  }
67  ?>
68
```

## Código Clase Planta

```
1  <¿php
2  /**
3   * @author César Cuéllar
4   * @version 1.0
5   * @created 20-oct-2013 07:47:30 a.m.
6   */
7  class Planta extends Empleado
8  {
9  private $deducciones;
10 private $sueldoBasico;
11 private $valorExtras;
12
13 //Constructor
14 public function Planta($identificacion,$nombre,$cargo)
15 {
16 //ejecutamos el constructor de la clase Padre Empleado
17 parent::__construct($identificacion,$nombre,$cargo);
18 }
19 public function calcularSalario()
20 {
21 $this->salario=$this->sueldoBasico + $this->valorExtras -
22 $this->deducciones;
23 }
24
25 public function getDeducciones()
26 {
27 return $this->deducciones;
28 }
29
30 public function getSueldoBasico()
31 {
32 return $this->sueldoBasico;
33 }
34
35 public function getValorExtras()
36 {
37 return $this->valorExtras;
38 }
39
40 public function getCargo()
```

```php
public function getCargo()
{
return $this->cargo;
}

public function getIdentificacion()
{
return $this->identificacion;
}

public function getNombre()
{
return $this->nombre;
}

public function getSalario()
{
return $this->salario;
}

/** @param newVal */
public function setDeducciones($newVal)
{
$this->deducciones = $newVal;
}

/** @param newVal */
public function setSueldoBasico($newVal)
{
$this->sueldoBasico = $newVal;
}

/** @param newVal */
public function setValorExtras($newVal)
{
$this->valorExtras = $newVal;
}

}
?>

```

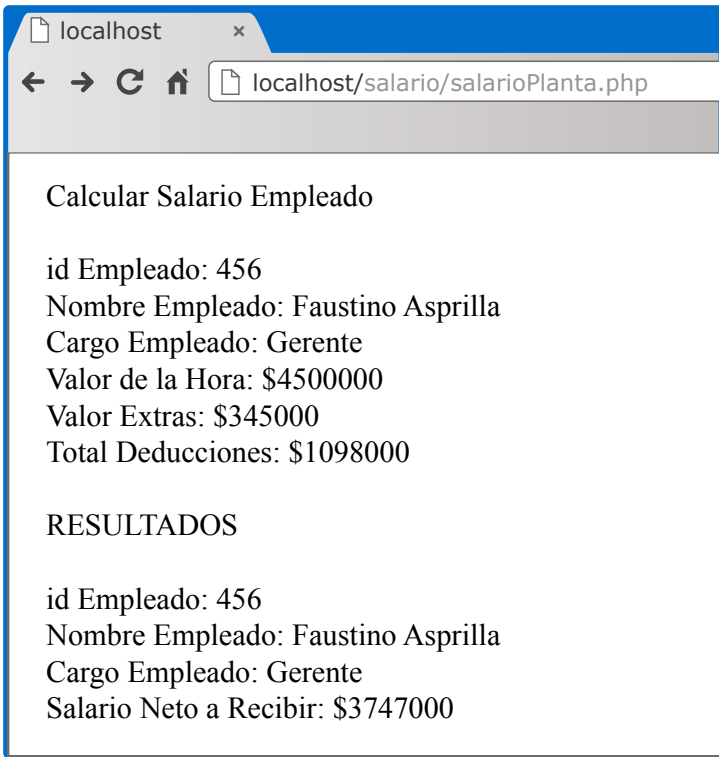
## Código de la Implementación Empleado Planta

```php
<¿php
include ('Empleado.php');
include "Planta.php";

echo "Calcular Salario Empleado";
echo "<br>";
//Datos de Entrada Empleado de Planta
$tipoEmpleado = "Planta";
$identificacion="456";
$nombre="Faustino Asptrilla";
$cargo = "Gerente";
$SueldoBasico = 4500000;
$valorExtras=345000;
$deducciones=1098000;

//Creamos el objeto
$objPlanta = new Planta($identificacion,$nombre,$cargo);

//modificamos atributos del empleado de planta
$objPlanta->setSueldoBasico($SueldoBasico);
$objPlanta->setValorExtras($valorExtras);
$objPlanta->setDeducciones($deducciones);

//imprimimos datos de entrada
echo "<br>id Empleado: " . $objPlanta->getIdentificacion();
echo "<br>Nombree Empleado: " . $objPlanta->getNombre();
echo "<br>Cargo Empleado: " . $objPlanta->getCargo();
echo "<br>Sueldo Basico: $" . $objPlanta->getSueldoBasico();
echo "<br>Valor Extras: $" . $objPlanta->getValorExtras();
echo "<br>Total Deducciones: $" . $objPlanta->getDeducciones();

echo "<br> <br> RESULTADOS <BR><BR>";

//calculam os el salario del empleado de planta

$objPlanta->calcularSalario();

//Mostramos resultados

echo "<br>id Empleado: " . $objPlanta->getIdentificacion();
echo "<br>Nombree Empleado: " . $objPlanta->getNombre();
echo "<br>Cargo Empleado: " . $objPlanta->getCargo();
echo "<br>Salario Neto a Recibir: $" . $objPlanta->getSalario();

?>
```

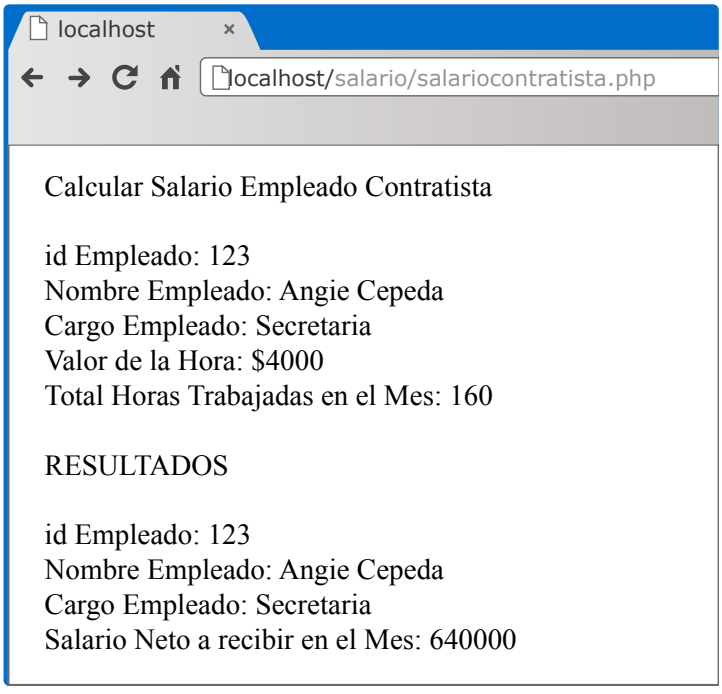
## Código Implementación Empleado Contratista

```php
<¿php
include ('Empleado.php');
include "Contratista.php";

//Datos de Entrada Empleado Contratista

echo "Calcular Salario Empleado Contratista ";
echo "<br>";
$tipoEmpleado = "Contratista";
$identificacion = "123";
$nombre = "Angie Cepeda";
$cargo = "Secretaria";
$totalHorasTrabajadas = 160;

//Creamos el objeto
$objContratista = new Contratista($identificacion,$nombre,$cargo);

//modificamos atributos del empleado de Contrato
$objContratista->setvalorHora(4000);
$objContratista->setTotalHoras($totalHorasTrabajadas);

//imprimimos datos de entrada
echo "<br>id Empleado: " . $objContratista->getIdentificacion();
echo "<br>Nombree Empleado: " . $objContratista->getNombre();
echo "<br>Cargo Empleado: " . $objContratista->getCargo();
echo "<br>Valor de la Hora: $" . $objContratista->getValorHora();
echo "<br>Total Horas Trabajas en el Mes: " . $objContratista->getTotalHoras();

echo "<br> <br> RESULTADOS <BR><BR>";

//calculam os el salario del empleado de Contrato
$objContratista->calcularSalario(4000,$totalHorasTrabajadas);

//Mostramos resultados
echo "<br>id Empleado: " . $objContratista->getIdentificacion();
echo "<br>Nombree Empleado: " . $objContratista->getNombre();
echo "<br>Cargo Empleado: " . $objContratista->getCargo();
echo "<br>Salario Neto a recibir en el Mes: " . $objContratista->getSalario();

?>
```

## Resultados Implementación Empleado Planta




## Resultado implementación Empelado Contratista



# RECURSOS BIBLIOGRÁFICOS

*Programación Orientada a Objetos en PHP. http://www.phpya.com.ar/poo/*

*Cobo, Angel. Gomez Patricia, Perez Daniel, Rocha Rocio. PHP y MySQL: Tecnología para el desarrollo de aplicaciones web.*

*Programación Orientada a Objetos en PHP. http://www.php.net/manual/es/language.oop5.php*

*Recursos en Videotutoriales Programación Orientada a Objetos en PHP. http://www.cesarcancino.com/?cat=2*

*Place, Enrique. Programación Orientada a Objetos en PHP 5. http://www.surforce.com*

# GLOSARIO

**POO:** Programación Orientada a Objetos.

**__construct:** Método Constructor de una clase en PHP.

**__destruct:** Método destructor de una clase en php.

**$this=**Variable que se utiliza en una clase para poder acceder a los métodos o atributos de la clase.

**class =** Palabra reservada utilizada en la definición de una clase.

**extends =** Palabra utilizada en la definición de una clase que hereda a otra clase.

| OBJETO DE APRENDIZAJE | PROGRAMACIÓN ORIENTADA A OBJETOS EN PHP |
|---|---|
| Desarrollador de contenido Experto temático | César Marino Cuéllar Chacón |
| Asesor Pedagógico | Rafael Neftalí Lizcano Reyes |
| Productores Multimedia | Manuel Francisco Silva Barrera<br>Luisa Fernanda Bolivar |
| Programador | Daniel Eduardo Martínez Díaz |
| Líder expertos temáticos | Ana Yaqueline Chavarro Parra |
| Líder línea de producción | Santiago Lozada Garcés |