



Análisis y desarrollo de software.

ID: 2669959



www.sena.edu.co

@SENAComunica

Instructor

Diego Fernando Calderón Silva
Correo: dfcalderon@sena.edu.co

4. Funciones



- **Include & Require**
- Ambas funciones sirven para añadir otros ficheros a nuestros script en PHP
 - Include_once:** Se incluye una única vez el archivo.
 - Require_once:** Se incluye una única vez el archivo.

Programación orientada a objetos (POO)



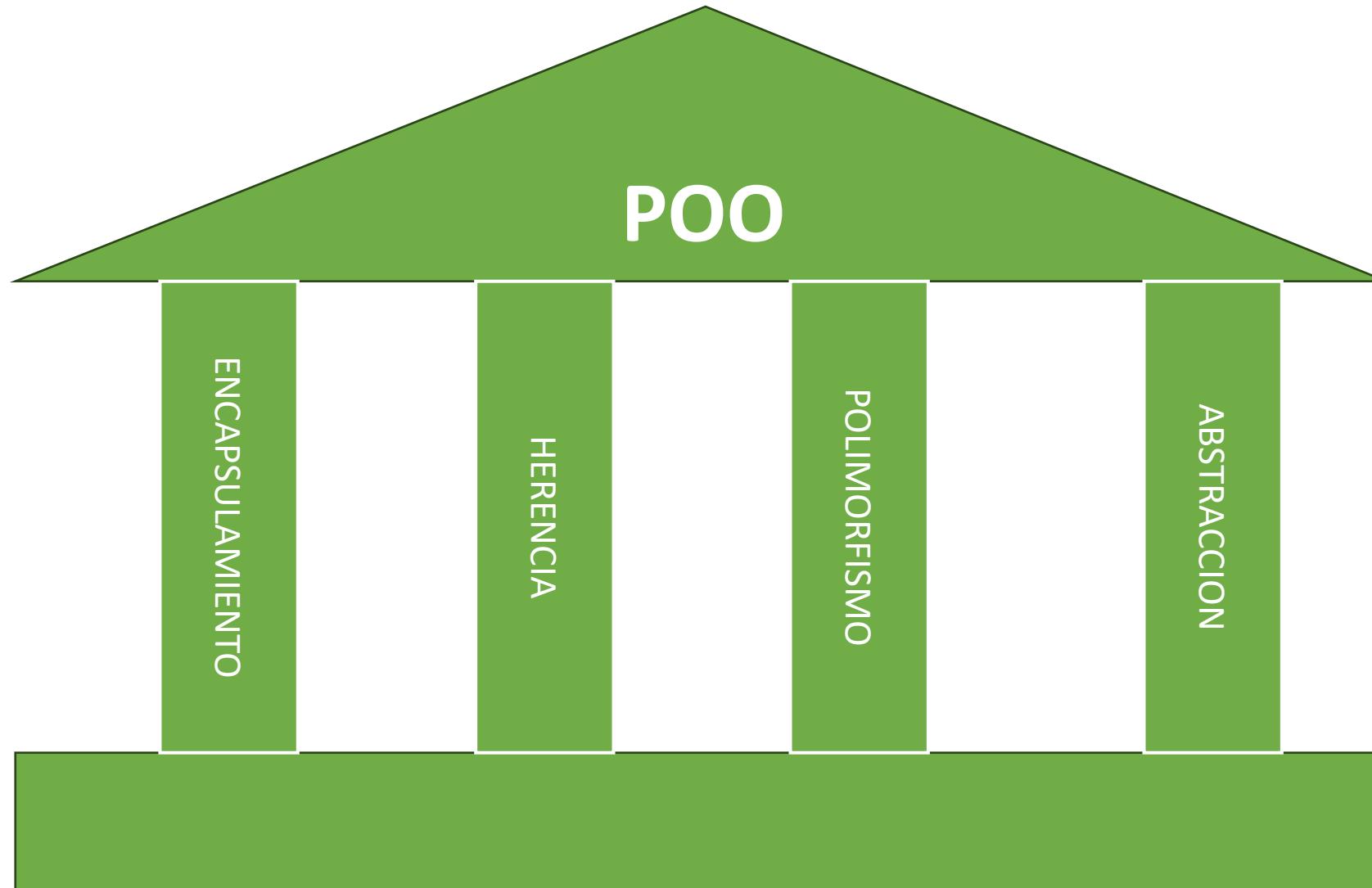
La programación orientada a objetos (POO) es el paradigma de programación más utilizado por la comunidad. Este tipo de programación se basa en la reutilización de código haciendo uso de clases, objetos y métodos. Se utiliza básicamente para darle una coherencia al código y hacerlo reutilizable.

Programación orientada a objetos (POO)



- La POO es un paradigma de programación (o técnica de programación) que utiliza objetos e interacciones en el diseño de un sistema.
- Paradigma: teoría cuyo núcleo central [...] suministra la base y modelo para resolver problemas [...] (Definición de la Real Academia Española, vigésimo tercera edición)
- Como tal, nos enseña un método -probado y estudiado- el cual se basa en las interacciones de objetos para resolver las necesidades de un sistema informático.
- Básicamente, este paradigma se compone de 4 pilares y diferentes características que veremos a continuación.

Programación orientada a objetos (POO)



Programación orientada a objetos (POO)



¿Qué es POO?

para definir POO podemos definir cada sigla con lo que sabes hasta ahora:

Programación: *Escribir código con el objetivo de darle ordenes al ordenador.*

Orientado a: *dirigido a.*

Objetos: modelo informático de un objeto real/ficticio.

Programación orientada a objetos (POO)



Objetos: Es una colección de datos con comportamientos asociados con el afán de representar en código a un objeto real o ficticio.

Propiedades: A la colección de datos que conforman un objeto se les conoce como: propiedades y sirven para describir cómo lucirán los objetos; por ejemplo: color, tamaño, peso, etc...

Métodos: A los comportamientos asociados a un objeto se les conoce como métodos, cada método es algo que nuestro objeto puede realizar: una función.



Programación orientada a objetos (POO)



Clase: En una clase se definen propiedades y métodos. Es el taller que contiene todas las herramientas necesarias para **crear objetos de un tipo específico** como un cuadro por lo que no sirve para crear un Automóvil. Si quisieras crear un Automóvil tendrías que crear una fábrica especialmente para eso.

Instancia: Crear un objeto es hacer una instancia. Si por ejemplo tienes definida una **clase Libro** (que contiene lo necesario para crear Libros) y creas un objeto partiendo de esa clase, estás haciendo una **instancia** de la **clase Libro**



Programación orientada a objetos (POO)

Ejemplo:

Objeto

- Goku
- Vegueta
- Gohan

Clase: Sayayin

Propiedades

- Terco
- Se convierten en SS
- Pelo puntudo
- SS de ojos verdes



Métodos

- Entrenar
- Dormir
- Jugar
- Comer

Programación orientada a objetos (POO)



CLASES: Una clase es un modelo que se utiliza para crear objetos que comparten un mismo comportamiento, estado e identidad por eje:



Class:OVNI

PROPIEDADES

- Gris
- Azul
- Redondo

METODOS

- Volar
- Abducciones
-

OBJETO
-Platillo volador

Programación orientada a objetos (POO)



CLASES: Una clase es un modelo que se utiliza para crear objetos que comparten un mismo comportamiento, estado e identidad por eje:



PROPIEDADES

- cantidadUno
- cantidadDos
- Resultado

Operaciones

METODOS

- Suma
- Resta
- Multiplicación
- División

OBJETO -Calculadora

Programación orientada a objetos (POO)



```
1 <?php
2
3 class Operacion{
4     //crear propiedades
5     public $cantidadUno = 0;
6     public $cantidadDos = 0;
7     public $resultado = 0;
8
9     //definir el metodo constructor
10    function __contruct($intCant1, $intCant2){
11        //para asignar valores hacemos lo siguiente
12        //propiedades
13        $this->cantidadUno = $intCant1;
14        $this->cantidadDos = $intCant2;
15    }
16
17    public function getSuma(){
18        $this->resultado = $this->cantidadUno + $this->cantidadDos;
19        return $this->resultado;
20    }
21    public function getResta(){
22        $this->resultado = $this->cantidadUno - $this->cantidadDos;
23        return $this->resultado;
24    }
25    public function getMultiplicacion(){
26        $this->resultado = $this->cantidadUno * $this->cantidadDos;
27        return $this->resultado;
28    }
29    public function getDivision(){
30        $this->resultado = $this->cantidadUno / $this->cantidadDos;
31        return $this->resultado;
32    }
33
34 } //end class Operacion
35
36 ?>
```

Programación orientada a objetos (POO)



```
index | < > index
index > No Selection
1 <?php
2 require_once ("classOperacion.php");
3
4 $hola = new Operacion(10,2);
5
6 $totalSuma = $hola->getSuma();
7 echo "El total de la suma es: ".$totalSuma;
8
```

Programación orientada a objetos (POO)



- **Abstracción:** Aislamiento de un elemento de su contexto. Define las características esenciales de un elemento(propiedades).
- **Encapsulamiento:** Reúne al mismo nivel de abstracción, a todos los elementos que puedan considerarse pertenecientes a una misma entidad.
- **Modularidad:** Característica que permite dividir una aplicación en varias partes más pequeñas (denominadas módulos), independientes unas de otras.

Programación orientada a objetos (POO)



- **Polimorfismo:** Es la capacidad que da a diferentes objetos, la posibilidad de contar con métodos, propiedades y atributos de igual nombre, sin que los de un objeto interfieran con el de otro.
- **Herencia:** Es la relación existente entre dos o más clases, donde una es la principal (padre) y otras son secundarias y dependen (heredan) de ellas (clases “hijas”), donde a la vez, los objetos heredan las características de los objetos de los cuales heredan.

Abstracción y Encapsulamiento



USUARIO



PROPIEDADES

- Nombre
- Email
- Tipo
- Clave

FUNCIONALIDADES

- REGISTRARSE
- Ver Perfil
- Cambiar clave

DIAGRAMA DE CLASES

Usuario
Nombre Email Tipo Clave
Registrarse() Ver_Perfil() Cambiar_Clave()

Abstracción y Encapsulamiento



```
ClassUsuario | < > ClassUsuario
ClassUsuario > No Selection

1 <?php
2
3     class Usuario{
4         //PROPIEDADES
5         private $strNombre;
6         private $strEmail;
7         private $strTipo;
8         private $strClave;
9         protected $strFechaRegistro;
10        static $strEstado = "Activo";
11
12    //METODOS
13    function __construct(string $nombre, string $email, string $tipo)
14    {
15        $this->strNombre = $nombre;
16        $this->strEmail = $email;
17        $this->strTipo = $tipo;
18        $this->strClave = rand();
19    }
20
21    }//End class usuario
22
23 ?>
24
```

Programación orientada a objetos (POO)



- **Modificadores de acceso:** Modificadores de acceso son palabras clave utilizadas en la programación orientada a objetos para especificar el nivel de visibilidad y accesibilidad de los miembros de una clase (atributos y métodos) desde otras partes del programa.

1. Public: Es un modificador de acceso que indica que el miembro de una clase es accesible desde cualquier parte del programa, tanto desde dentro de la clase como desde fuera de ella. Los miembros públicos se pueden acceder y utilizar libremente por otros objetos y clases.

2. Private: Es un modificador de acceso que indica que el miembro de una clase solo es accesible desde dentro de la propia clase. Los miembros privados no se pueden acceder o utilizar directamente desde otras clases u objetos. Su uso principal es encapsular y ocultar la implementación interna de una clase, lo que mejora la seguridad y el control sobre los datos y comportamientos internos.

3. Protected: Es un modificador de acceso similar al privado, pero permite que los miembros de una clase sean accesibles también desde las clases derivadas (subclases) de esa clase. Esto significa que los miembros protegidos pueden ser utilizados en la clase base y en sus subclases, pero no desde otras clases que no sean derivadas de ella.

Programación orientada a objetos (POO)



- **Modificadores de acceso:**

4. Static: No es estrictamente un modificador de acceso, sino un modificador que afecta el ámbito y comportamiento de un miembro de clase. Cuando se aplica el modificador static a un miembro (variable o método), se asocia con la clase en lugar de con las instancias individuales de la clase. Esto significa que se puede acceder al miembro estático sin crear una instancia de la clase. Los miembros estáticos se comparten entre todas las instancias de la clase y se pueden acceder mediante el nombre de la clase en lugar de a través de un objeto específico

Modificadores y métodos de acceso(Getter y Setter)

```
1  <?php
2
3 usages
4 class Usuario{
5     //la abstraccion por que definimos las propiedades
6     //propiedad
7     3 usages
8     private $strName;
9     3 usages
10    private $strEmail;
11    2 usages
12    private $strTipo;
13    3 usages
14    private $strClave;
15    2 usages
16    protected $strFechaRegistro;
17    2 usages
18    static $strEstado = "Activo";//se puede acceder sin instanciar
19    //metodo constructor
20    2 usages
21    function __construct(string $nombre, string $email, string $tipo){
22        $this -> strName = $nombre;
23        $this -> strEmail = $email;
24        $this -> strTipo = $tipo;
25        $this -> strClave = rand();
26        $this -> strFechaRegistro = date( format: 'Y-m-d H:m:s' );
27    }
28
29 } 1 usage
```

```
1  <?php
2
3 require_once "classUsuario.php";
4
5 $objUsuario1 = new Usuario( nombre: "diego", email: "hola@ingo.com", tipo: "admin");
6
7 //echo $objUsuario1->strName."\n";//si fuera publico asi lo podria llamar
8 echo $objUsuario1->getNombre()."\n"; //asi puedo acceder al metodo privado llamando a
9 la funcion getNombre
10
11 //encapsulamiento no permite acceder desde un lugar que no sea la clase donde se creo,
12 esto mismo pasa con protected y static
13 //echo $objUsuario1->srtClave;
14 //echo $objUsuario1->strTipo."\n";
15 echo $objUsuario1->getTipo()."\n";
16
17 //nos estamos dirigiendo directamente a la propiedad
18 //entonces no se necesita instanciar una clase para hacer uso de las declaraciones ya
19 que le corresponde a la clase no a los objetos que se estan creando
20 echo Usuario::$strEstado."\n";
21
22 //mostrar la informacion del usuario
23
24 echo $objUsuario1->getPerfil();
25
26 echo "\n\n";
27 //Voy a crear un nuevo metodo llamado andrea
28
29 $objAndrea = new Usuario( nombre: "Andrea", email: "andrea@info.com", tipo: "vendedor");
```

Modificadores y métodos de acceso(Getter y Setter)

```
//constructor  
  
3 usages  
function __construct(string $nombre, string $documento){  
    $this -> strNombre = $nombre;  
    $this -> strDocumento = $documento;  
  
}  
  
1 usage  
public function getNombre():string{  
    return $this->strNombre;  
}  
  
no usages  
public function getDocumento():string{  
    return $this -> strDocumento;  
}  
  
1 usage  
public function setPlaca(string $placa){  
    $this -> strPlaca = $placa;  
}  
  
1 usage  
public function perfilUsuario(){  
    echo "LOS DATOS DEL USUARIO ".strtoupper($this->strNombre)."\n";  
    echo "Documento ".$this->strDocumento."\n";  
    echo "Placa ".$this->strPlaca."\n";  
}  
}//FIN CLASS USUARIO
```

```
▲ 9 ▼ 7 ^ v 1 <?php  
2  
3 require_once ("classParqueadero.php");  
4  
5 $placa = readline( prompt: "Ingrese el numero de placa del usuario: \n");  
6 $objUsuario = new Usuario("Diego", "1094914260");  
7 //asi se muestran los publicos  
8 //echo "El documento del usuario es: ".$objUsuario->strDocumento." y su nombr  
9  
10 $objUsuario->setPlaca($placa);  
11  
12 echo $objUsuario->perfilUsuario();  
13 //echo "El usuario ".$objUsuario->getNombre()." con documento ".$objUsuario->
```

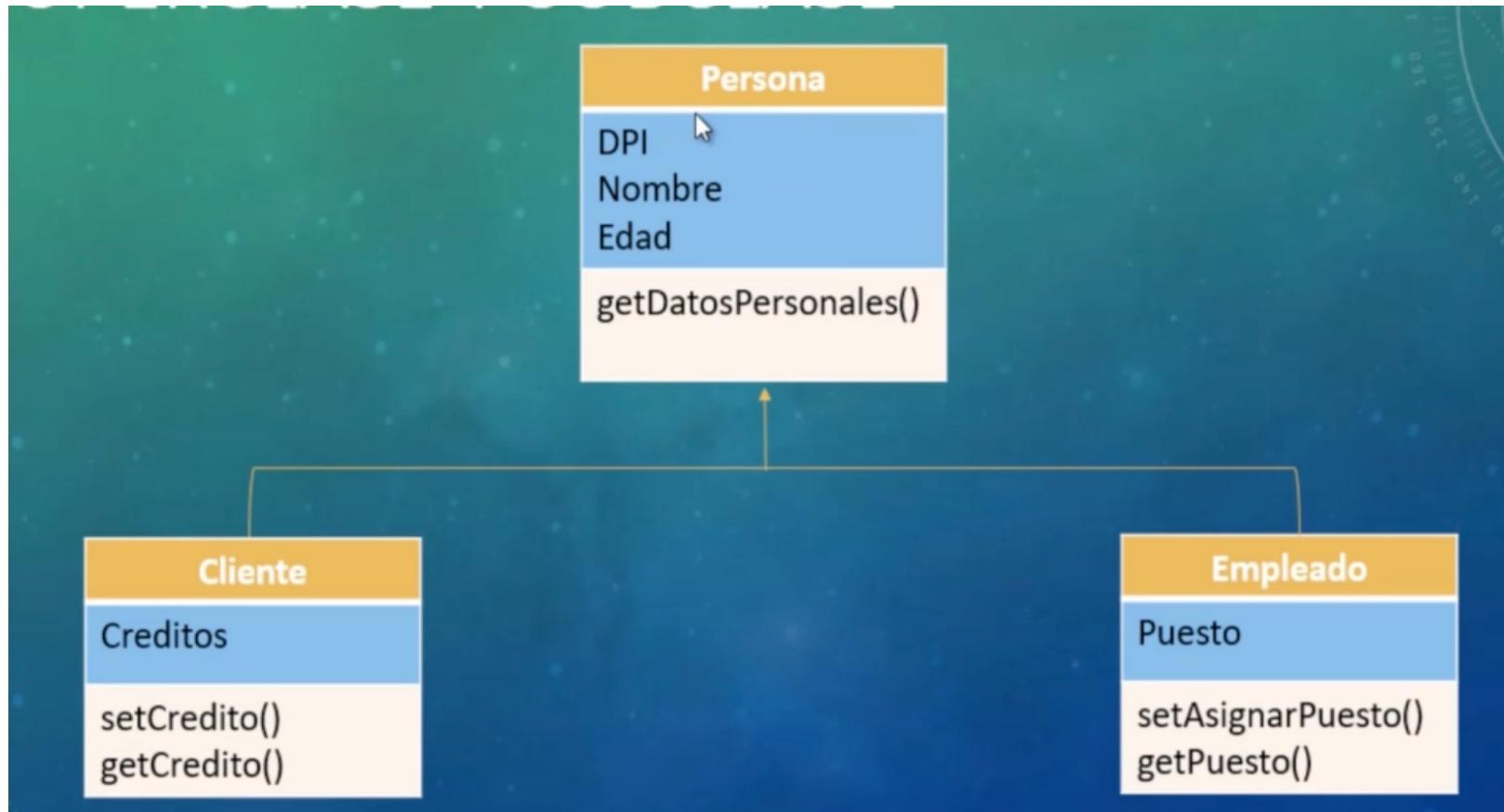
Herencia, super clase y subclase



Los objetos pueden heredar propiedades y métodos de otros objetos. Para ello, PHP permite la “extensión” (herencia) de clases, cuya característica representa la relación existente entre diferentes objetos. Para definir una clase como extensión de una clase “padre” se utiliza la palabra clave “extends”.

```
<?php  
  
class clasePadre {  
    ...  
}  
class claseHija extends clasePadre {  
    /* esta clase hereda todos los métodos y propiedades de  
    la clase madre NombreDeMiClaseMadre */  
}  
?  
?
```

Herencia, super clase y subclase



Herencia, super clase y subclase



The screenshot shows a code editor with multiple tabs open, but the central tab, "ClassPersona.php", is active and highlighted with a yellow bar at the top. The code within this tab defines a class named "Persona". The class has four public attributes: \$intDpi, \$strNombre, \$intEdad, and \$mensaje. It also contains a constructor "__construct" that initializes these attributes with their respective parameters. A method "getDatosPersonales" is defined to return a string containing the person's data in a specific format. The code editor provides syntax highlighting and includes annotations such as "4 usages 6 inheritors" for the class itself and "2 usages" for each attribute.

```
<?php
class Persona{
    public $intDpi;
    public $strNombre;
    public $intEdad;
    public $mensaje;
    function __construct (int $dpi, string $nombre, int $edad)
    {
        $this->intDpi = $dpi;
        $this->strNombre = $nombre;
        $this->intEdad = $edad;
    }
    public function getDatosPersonales()
    {
        $datos = "
            <h2>DATOS PERSONALES</h2>
            DPI: {$this->intDpi}<br>
            Nombre: {$this->strNombre}<br>
            Edad: {$this->intEdad}<br>
        ";
        return $datos;
    }
}
//End class persona
```

Herencia, super clase y subclase



The screenshot shows a code editor with multiple tabs at the top: classUsuarios.php, usuario.php, ClassPersona.php, ClassCliente.php (which is selected), and ClassEmpleado.php. The main pane displays the following PHP code:

```
<?php
require_once ("ClassPersona.php");

class Cliente extends Persona{
    protected $fltCredito;
    function __construct(int $dpi, string $nombre, int $edad){
        parent::__construct($dpi, $nombre, $edad);
    }
    public function setCredito(string $credito){
        $this->fltCredito = $credito;
    }
    public function getCredito():float
    {
        return $this->fltCredito;
    }
} //End class empleado
```

Herencia, super clase y subclase



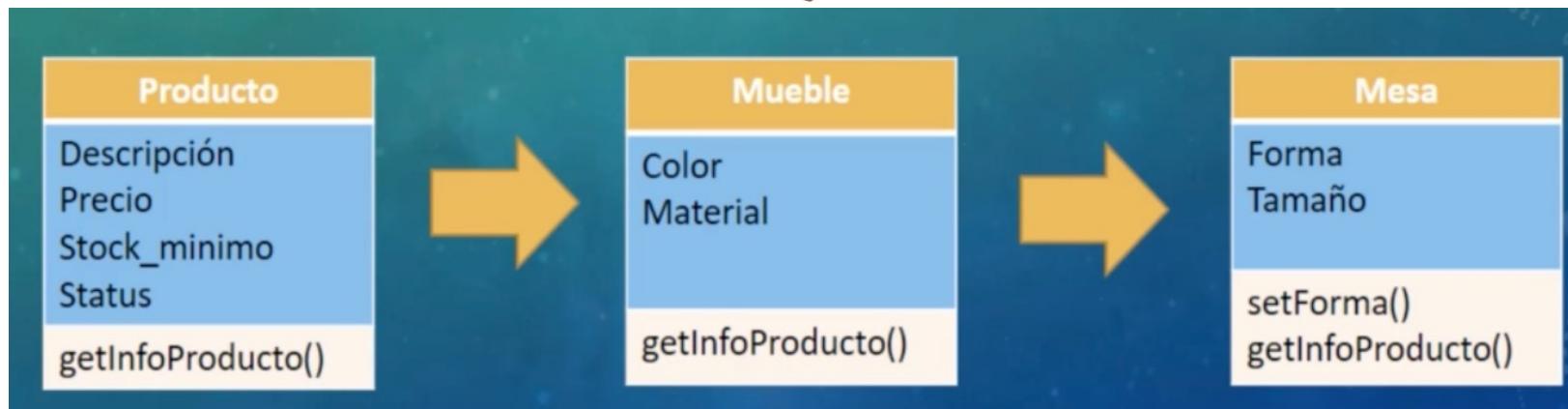
```
class Usuarios.php x usuario.php x ClassPersona.php x ClassCliente.php x ClassEmpleado.php x
1 <?php
2     require_once ("ClassPersona.php");
3     1 usage
4     class Empleado extends Persona{
5         2 usages
6         protected $strPuesto;
7         1 usage
8         function __construct(int $dpi, string $nombre, int $edad){
9             parent::__construct($dpi, $nombre, $edad);
10            }
11            1 usage
12            public function setPuesto(string $puesto){
13                $this->strPuesto = $puesto;
14            }
15            1 usage
16            public function getPuesto():string
17            {
18                return $this->strPuesto;
19            }
20        } //End class empleado
21
22    ?>
```

Herencia, super clase y subclase



```
class Usuarios.php × PHP usuario.php × ClassPersona.php × ClassCliente.php × ClassEmpleado.php × PHP empresa.php ×
1 <?php
2
3     require_once ("ClassEmpleado.php");
4     require_once ("ClassCliente.php");
5     require_once ("ClassPersona.php");
6
7     $objEmpleado = new Empleado( dpi: 78978, nombre: "Andres Pérez", edad: 25);
8     $objEmpleado->setPuesto( puesto: "Administrador");
9
10
11    echo $objEmpleado->getDatosPersonales();
12    echo "Puesto:". $objEmpleado->getPuesto();
13
14    $objCliente = new Cliente( dpi: 434543, nombre: "Elena Castillo", edad: 20);
15    $objCliente->setCredito( credito: 1000);
16
17
18    echo $objCliente->getDatosPersonales();
19    echo "Créditos:". $objCliente->getCredito();
20
21
22
23 ?>
```

Polimorfismo(Redefinición de métodos y propiedades)



Sistema de Gestión de Pedidos en un Restaurante

Descripción del Ejercicio:

En este ejercicio, se espera que desarrolle un sistema de gestión de pedidos para un restaurante utilizando programación orientada a objetos en PHP. El sistema permitirá a los clientes realizar pedidos desde sus mesas, que serán enviados a la cocina para su preparación. Además, se implementará una caja registradora para cobrar los pedidos.

Requisitos:

1. Clases Principales:

- `Restaurante`: Representa el restaurante y gestiona las mesas, la caja registradora y el chef.
- `Mesa`: Modela una mesa en el restaurante, permite a los clientes agregar pedidos y cobrar.
- `Cliente`: Define a un cliente que realiza pedidos.
- `Pedido`: Representa un pedido con varios ítems.
- `Item`: Describe un ítem del menú con nombre y precio.
- `Menu`: Almacena los ítems disponibles en el menú.
- `CajaRegistradora`: Gestiona las ventas y el cálculo de ingresos.
- `Chef`: Prepara y acepta pedidos de la cocina.

Desarrolle



Sistema de Gestión de Pedidos en un Restaurante

2. Funcionalidades:

- Mesa:

- Puede agregar clientes a la mesa.
- Puede realizar pedidos a través del cliente.
- Puede cobrar los pedidos y registrar el pago en la caja registradora.

- Cliente:

- Puede realizar pedidos seleccionando ítems del menú.
- Puede ver el total de su pedido.

- Pedido:

- Debe contener una lista de ítems pedidos.
- Debe calcular el total del pedido.

- CajaRegistradora:

- Debe mantener un registro de las ventas totales.

- Chef:

- Debe preparar y aceptar pedidos de la cocina.

Desarrolle



Sistema de Gestión de Pedidos en un Restaurante

3. Clases Adicionales (Opcionales):

- Pueden agregar subclases de `Cliente` para representar diferentes tipos de clientes, como `ClienteRegular` y `ClienteVIP`, con comportamientos específicos.

4. Instrucciones

- Crea todas las clases y métodos necesarios para implementar las funcionalidades mencionadas.
- Define un menú con una lista de ítems y sus precios.
- Simula la interacción entre clientes, mesas, la caja registradora y el chef.
- Registra las ventas totales en la caja registradora.
- Asegúrate de utilizar principios de herencia y polimorfismo cuando sea necesario.
- Realiza pruebas exhaustivas del sistema para garantizar su correcto funcionamiento.

Con Este ejercicio se une oportunidad para que practiquen programación orientada a objetos en PHP mientras desarrollan un sistema de gestión de pedidos de restaurante completo. Les deseo mucho éxito en su desarrollo y aprendizaje.



G R A C I A S

Línea de atención al ciudadano: 01 8000 910270
Línea de atención al empresario: 01 8000 910682



www.sena.edu.co