

# PROGRAMACIÓN II



**SEMANA 09**  
**CLASES Y OBJETOS**

Eric Gustavo Coronel Castillo  
[ecoronel@uch.edu.pe](mailto:ecoronel@uch.edu.pe)  
[gcoronelc.blogspot.com](http://gcoronelc.blogspot.com)

# CLASES Y OBJETOS

En este capítulo se desarrolla los primeros conceptos de la POO. Veremos los conceptos de clase, atributo y operación usando notación UML; luego su implementación utilizando PHP, donde se conoce como clase, campo y método.

## Índice

1	DEFINICIÓN .....	4
2	USO DE LA FUNCIÓN ARRAY() .....	¡ERROR! MARCADOR NO DEFINIDO.
3	USO DE FOREACH().....	¡ERROR! MARCADOR NO DEFINIDO.

## 1 Definición de una Clase

---

### 1.1 Definición de una Clase

Una clase define un tipo de objeto en particular.

Por ejemplo, en un sistema de ventas la clase **Factura** define a todas las facturas que la empresa emite.

Su sintaxis es:

```
[ModificadorClase] class NombreClase {  
  
    // Campos  
  
    // Métodos  
  
}
```

El **ModificadorClase** puede tomar los valores que se resumen en el siguiente cuadro:

Palabra Clave	Descripción
<b>abstract</b>	Define una clase como abstracta, quiere decir que no se podrán crear instancias de la clase definida como abstracta.
<b>final</b>	Si una clase se define como <b>final</b> , entonces la clase no podrá ser extendida, quiere decir que no se podrán crear subclases desde una clase <b>final</b> .

### Ejemplo 1

```
class Factura {  
  
    // Campos  
    private $numero;  
    private $importe;  
  
    // Métodos  
    public function grabar( . . . ) {  
  
        . . .  
    }  
}
```

```

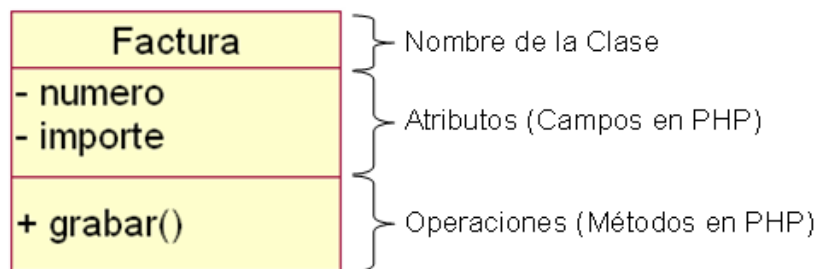
    ...
}

} // Factura

```

## 1.2 Representación UML de una Clase

Para tener una representación gráfica de una clase se utiliza UML (Unified Modeling Language), tal como se ilustra en la Figura 1.



**Figura 1** Representación UML de una clase.

## 1.3 Creación de Objetos

El operador `new` se utiliza para crear un objeto de un tipo de clase específica, es decir, que asigne memoria para el objeto.

Para utilizar el operador `new` tenemos dos sintaxis.

### Sintaxis 1

```

private $nombreDeVariable;
$nombreDeVariable = new NombreDeClase();

```

En la primera instrucción se define la variable que apuntará a un objeto que se crea en la segunda instrucción.

### Sintaxis 2

```

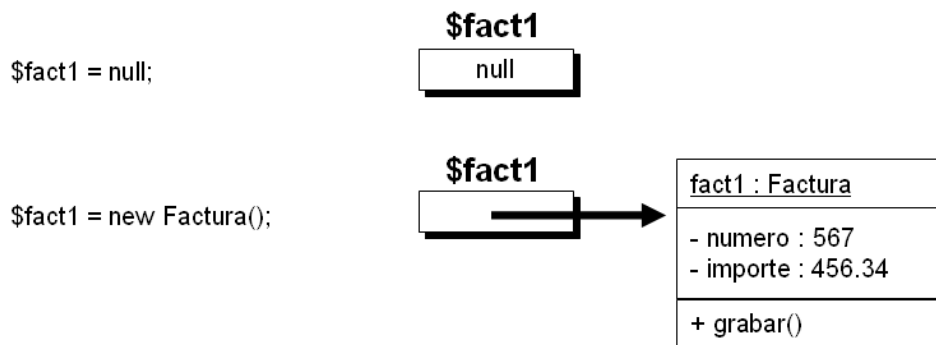
private $nombreDeVariable = new NombreDeClase();

```

En este caso tanto la creación de la variable y la creación del objeto se realizan en la misma instrucción.

### Instrucción

### Efecto

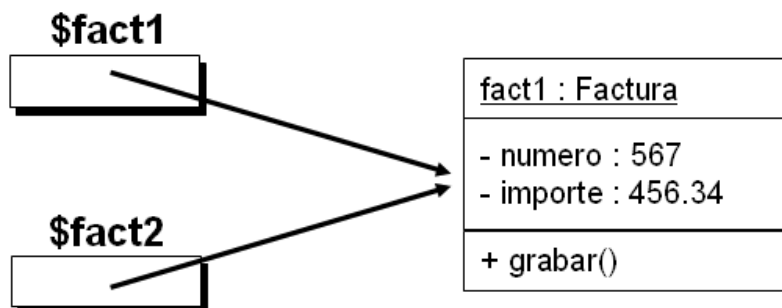


**Figura 2** Instanciación de una clase (Creación de un objeto).

## 1.4 Asignación de Objetos

Cuando creamos un objeto internamente existe un puntero al que no tenemos acceso, pero cuando asignamos objetos, lo que realmente estamos asignando son las direcciones de memoria donde están definidos los objetos, como se explica en la Figura 3.

```
$fact1 = new Factura();
$fact2 = $fact1;
```



**Figura 3** Asignación de objetos.

## 2 Trabajando con Campos

---

### 2.1 Definición

Los campos definen los datos de un objeto; cuando definimos un campo debemos declararlo y establecer su visibilidad.

La visibilidad determina desde que parte del código se puede acceder al campo.

**Sintaxis:**

```
[TipoAcceso] [ModificadorCampo] $nombreCampo [= ValorInicial];
```

El **TipoAcceso** puede tomar los valores que se resume en el siguiente cuadro:

Palabra Clave	Descripción	UML
<b>public</b>	Indica que el campo es de acceso público. El acceso se realiza a través de una instancia del objeto.	+
<b>protected</b>	Indica que solo se puede acceder al campo desde métodos de la misma clase y clases derivadas (subclases)	#
<b>private</b>	Indica que solo se puede acceder al campo desde métodos de la misma clase.	-

El **ModificadorCampo** puede tomar los valores que se resumen en el siguiente cuadro:

Palabra Clave	Descripción
<b>static</b>	El campo pertenece a la clase, no a los objetos creados a partir de ella.

El **ValorInicial** permite inicializar la variable con un valor.

De acuerdo a la forma en que se especifica un atributo, objetos de otras clases tienen distintas posibilidades de accederlos, tal como se resume en el siguiente cuadro:

Acceso Desde	private	protected	public
La propia clase	Si	Si	Si
Subclase	No	Si	Si

## 2.2 Ocultando los Datos

Uno de los fundamentos de la programación orientada a objetos es que el usuario solo debe tener acceso a los datos que le son de su interés, y del modo que le corresponde, por ejemplo sólo lectura, sólo escritura, ó ambos.

Para conseguir esto se debe declarar los atributos como privados, y se debe implementar métodos para acceder a ellos. Existe un estándar para definir estos métodos, por ejemplo, si el campo es **\$nombre** los métodos son:

- **setNombre()** Este método permite asignar un valor al campo.
- **getNombre()** Este método permite leer el valor del campo.

Como puede apreciar existen dos prefijos, el prefijo **set** que se utiliza para asignar un valor al campo y el prefijo **get** para leer el valor del campo; de esta manera podemos seleccionar si se implementa el método **set**, **get**, ó ambos, y restringir el nivel de acceso a los datos.

Otra posibilidad que nos da la implementación de estos métodos es de poder agregar funcionalidad que puede servir para verificar, por ejemplo, si el dato que se está signando es correcto o no.

## Ejemplo 2

En este ejemplo se ilustra el uso de los métodos **set** y **get**.

Archivo: Factura.php

---

```
<?php
class Factura {

    // Campos

    private $numero;

    // Métodos set y get

    public function getNumero() {
        return $this->numero;
    }

    public function setNumero($numero) {
        $this->numero = $numero;
    }

}
?>
```

Como puede apreciar en el código, el método **setNumero()** se utiliza para asignar un valor al campo **\$numero**, mientras que el método **getNumero()** se utiliza para leer su valor.

Es posible agregar lógica en cualquiera de los dos métodos, o en ambos si el caso lo amerita.



## 3 Trabajando con Métodos

### 3.1 Definición

Los métodos definen el comportamiento de un objeto de una clase concreta y tienen las siguientes características:

- Agrupan las responsabilidades del objeto (competencias).
- Describen sus acciones.

Las acciones realizadas por un objeto son consecuencia directa de un estímulo externo (un mensaje enviado desde otro objeto) y dependen del estado del objeto.

El estado y comportamiento de un objeto están relacionados. Por ejemplo, un avión no puede aterrizar (acción) sino está en vuelo (estado).

**Sintaxis:**

```
[TipoAcceso] [ModificadorMétodo]
function NombreMétodo ( [ ListaDeParámetros ] ) {

    // Cuerpo de método

    [ return expresión; ]

}
```

El **TipoAcceso** puede tomar los valores que se resumen en el siguiente cuadro:

Palabra Clave	Descripción	UML
<b>public</b>	Indica que el método es de acceso público. El acceso se realiza a través de una instancia del objeto.	+
<b>protected</b>	Indica que solo se puede acceder al método desde métodos de la misma clase y clases derivadas (subclases)	#
<b>private</b>	Indica que solo se puede acceder al método desde métodos de la misma clase.	-
<b>sin especificar</b>	Por defecto se asume que el nivel de acceso es público.	

El **ModificadorMétodo** puede tomar los valores que se resumen en el siguiente cuadro:

Palabra Clave	Descripción
<b>static</b>	El método pertenece a la clase, no a los objetos creados a partir de ella.
<b>final</b>	El método es definido como método definitivo, esto quiere decir que no se puede redefinir en una subclase.
<b>abstract</b>	Determina que el método no se implementa en la clase, su implementación será en las clases heredadas o subclases.

Los parámetros son opcionales, pero si son necesarios se deben especificar siguiendo las reglas estudiadas para los parámetros de las funciones de usuario.

Sólo para el caso en que el parámetro sea un objeto se debe especificar el tipo de objeto (Clase), esto se conoce como **Type Hinting**, se utiliza la siguiente sintaxis:

```
nombreMétodo( NombreClase $parámetro, ... )
```

Los parámetros son variables que reciben valores de los argumentos que se le pasan al método cuando éste es invocado.

Los métodos que necesitan retornar un valor a la instrucción que realiza la llamada deben utilizar la instrucción **return**; a continuación tenemos su sintaxis:

```
return expresión;
```

Donde expresión representa el valor que retorna el método.

## 4 Proyectos Resueltos

---

### 4.1 Encontrar el Interés Compuesto

#### 4.1.1 Requerimientos del Software

Una institución financiera necesita de un programa que le permita encontrar el importe que deben pagar sus clientes por los préstamos que realiza, se sabe que se trata de un interés compuesto, capitalizable mensualmente.

La fórmula que debe aplicarse es:

$$M = C(1 + i)^n$$

Donde:

- C : Capital
- i : Tasa de interés por periodo, por ejemplo puede ser mensual
- n : Número de periodos
- M : Importe acumulado en el número de periodos

#### 4.1.2 Abstracción

A continuación tenemos algunas alternativas de abstraer el problema.

##### Caso 01

En este caso los atributos son públicos (visibilidad publica) por lo tanto se puede acceder de manera directa desde cualquier otra clase u objeto.

El método **calcularImporte()** hace el cálculo en función al valor de los campos capital, interés y periodo.

Banco
+ capital : double
+ interes : double
+ periodo : int
+ calcularImporte()

##### Caso 02

Banco
+ calcularImporte(capital : double, interes : double, periodos : int) : Double

En este caso tenemos un solo método que tiene tres parámetros, a través de los cuales recibe los datos para hacer el cálculo y retorna el resultado correspondiente.

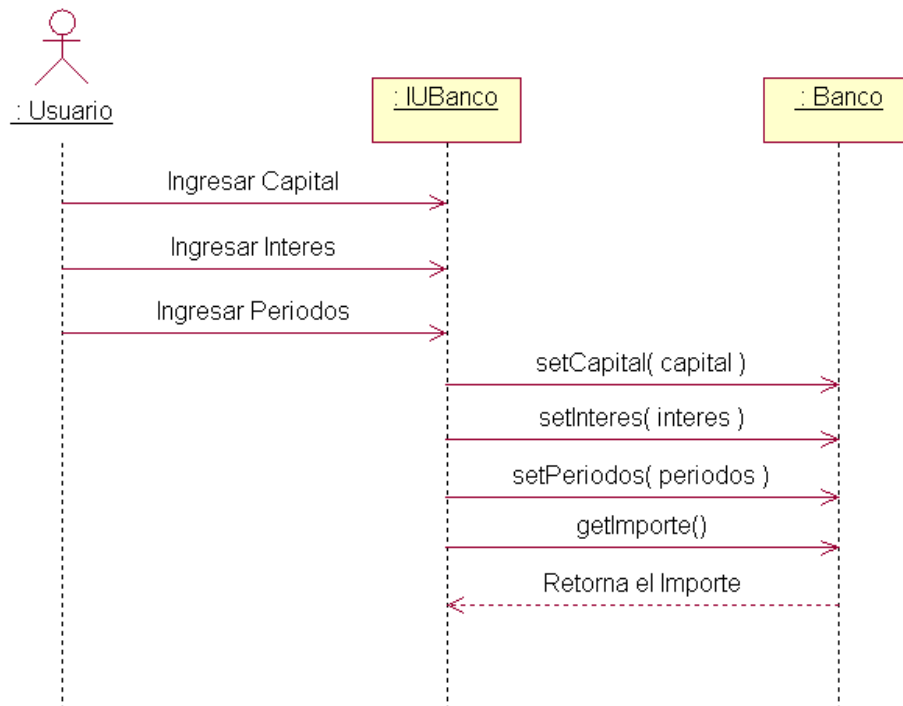
### Caso 03

En este caso, los atributos son privados y se accede a ellos a través de los métodos **set** y **get**.

Para obtener el importe se debe utilizar el método **getImporte()**, el cálculo se realiza en base a los campos respectivamente.

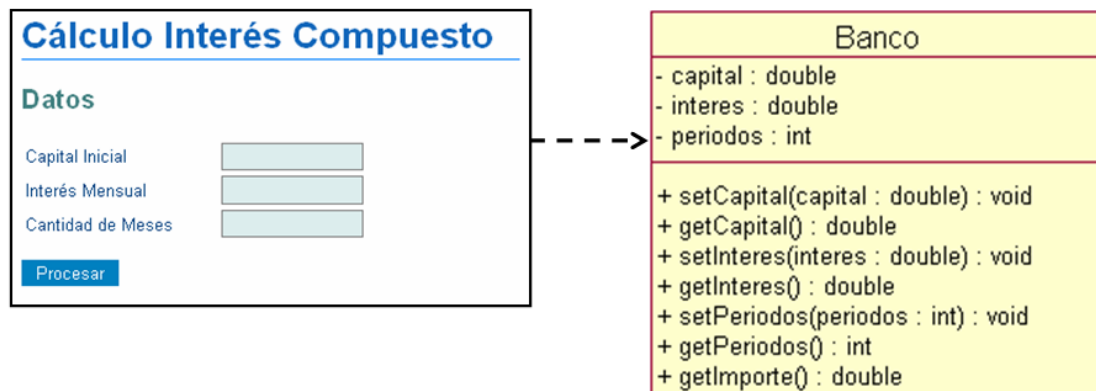
Banco
- capital : double - interes : double - periodos : int
+ setCapital(capital : double) : void + getCapital() : double + setInteres(interres : double) : void + getInteres() : double + setPeriodos(periodos : int) : void + getPeriodos() : int + getImporte() : double

#### 4.1.3 Diagrama de Secuencia



#### 4.1.4 Diagrama de Clases

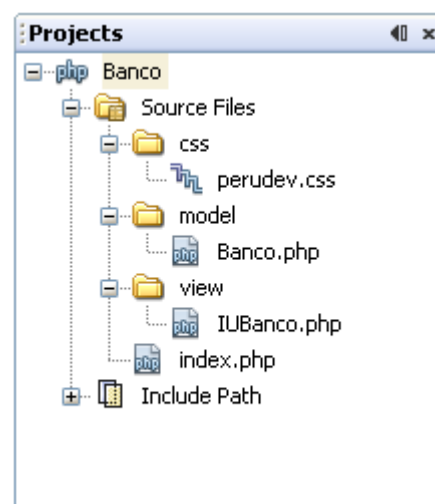
En este caso tenemos un programa PHP que hará uso de la clase Banco, y gráficamente lo representamos de la siguiente manera:



#### 4.1.5 Estructura del Proyecto en NetBeans

Como se puede apreciar en el grafico de la derecha, tenemos tres carpetas, en la carpeta **css** tenemos la hoja de estilos, en la carpeta **model** ubicamos la clase **Banco**, y en la carpeta **view** el programa **IUBanco.php**.

En la raíz del proyecto tenemos el programa **index.php**, éste es el que inicia la ejecución del proyecto, y en este caso es hacer un redireccionamiento al programa **IUBanco.php**.



#### 4.1.6 Codificación de Clase Banco

Archivo: Banco.php

```
<?php
class Banco {

    // Campos

    private $capital = 0.0;
    private $interes = 0.0;
    private $periodos = 0.0;
```

```
// Métodos

public function getCapital() {
    return $this->capital;
}

public function setCapital($capital) {
    $this->capital = $capital;
}

public function getInteres() {
    return $this->interes;
}

public function setInteres($interes) {
    $this->interes = $interes;
}

public function getPeriodos() {
    return $this->periodos;
}

public function setPeriodos($periodos) {
    $this->periodos = $periodos;
}

public function getImporte(){
    $importe = $this->getCapital() *
        pow(1 + $this->getInteres(), $this->getPeriodos());
    return round($importe,2);
}

}
?>
```

#### 4.1.7 Codificación de la Interfaz de Usuario IUBanco

Archivo: IUBanco.php

```
<?php
require_once '../model/Banco.php';
$formulario = TRUE;
if( isset( $_POST["procesar"] ) ) {
    $formulario = FALSE;
    $banco = new Banco();
    $banco->setCapital($_POST["capital"]);
    $banco->setInteres($_POST["interes"]);
    $banco->setPeriodos($_POST["meses"]);
}
?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
    <link rel="stylesheet" type="text/css" href="../css/perudev.css"/>
    <title></title>
  </head>
  <body>
    <h1>Cálculo Interés Compuesto</h1>
    <?php if( $formulario) { ?>
    <div id="datos">
      <h2>Datos</h2>
      <form name="form1" method="post" action="IUBanco.php">
        <table width="286">
          <tr>
            <td width="129">Capital Inicial</td>
            <td width="145">
              <input name="capital" type="text" class="fieldEdit"
                id="capital" size="10" maxlength="10">
            </td>
          </tr>
          <tr>
            <td>Interés Mensual</td>
            <td>
              <input name="interes" type="text" class="fieldEdit"
                id="interes" size="10" maxlength="5">
            </td>
          </tr>
        </table>
      </form>
    </div>
  </body>
</html>
```

```

        <td>Cantidad de Meses</td>
        <td>
            <input name="meses" type="text" class="fieldEdit"
                id="meses" size="10" maxlength="5">
        </td>
    </tr>
</table>
<input name="procesar" type="submit" class="button"
    id="procesar" value="Procesar">
</form>
</div>
<?php } else { ?>
<div id="resultado">
    <h2>Resultado</h2>
    <table width="300">
        <tr>
            <td width="136">Capital Inicial</td>
            <td width="152"><?php echo($banco->getCapital()) ?></td>
        </tr>
        <tr>
            <td>Interés Mensual</td>
            <td><?php echo($banco->getInteres()) ?></td>
        </tr>
        <tr>
            <td>Cantidad de Meses</td>
            <td><?php echo($banco->getPeriodos()) ?></td>
        </tr>
        <tr>
            <td>Importe Total</td>
            <td><?php echo($banco->getImporte()) ?></td>
        </tr>
    </table>
    <p><a href="IUBanco.php">Otro Cálculo</a></p>
</div>
<?php } ?>
</body>
</html>

```



### 4.1.8 Codificación del Programa de Inicio

Archivo: index.php

```
<?php header("location: view/IUBanco.php") ?>
```

### 4.1.9 Ejecución del Proyecto

Cuando ejecutamos el proyecto tenemos la siguiente interfaz:

## Cálculo Interés Compuesto

### Datos

Capital Inicial	<input type="text" value="1000"/>
Interés Mensual	<input type="text" value="0.025"/>
Cantidad de Meses	<input type="text" value="12"/>

Luego de ingresar datos de prueba y hacer clic en el botón **Procesar** obtenemos el siguiente resultado:

## Cálculo Interés Compuesto

### Resultado

Capital Inicial	1000
Interés Mensual	0.025
Cantidad de Meses	12
Importe Total	1344.89

[Otro Cálculo](#)

## 4.2 Los Genios

### 4.2.1 Requerimientos del Software

El colegio "**Los Genios**" está haciendo una reforma en los procesos de aprendizaje de sus alumnos; está utilizando programas de computadora para que los mismos alumnos dirijan su aprendizaje.

El colegio requiere en estos momentos un programa en PHP para que sus alumnos de los primeros grados aprendan las cuatro operaciones en forma de adivinanza, el programa debe felicitar al alumno cuando ingrese la respuesta correcta.

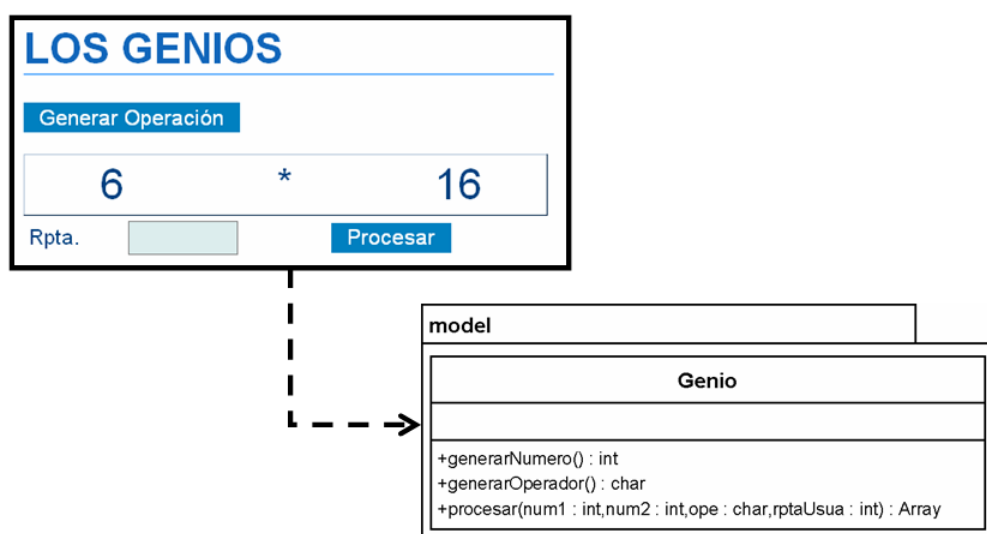
### 4.2.2 Abstracción

La solución planteada considera una clase de nombre **Genio** con tres métodos que se describen a continuación:

Método	Descripción
<b>generarNumero()</b>	Genera un número en forma aleatoria entre 1 y 20.
<b>generaOperador()</b>	Genera en forma aleatoria un operador.
<b>procesar()</b>	Procesa los datos y retorna el resultado en un arreglo.

### 4.2.3 Diagrama de clases

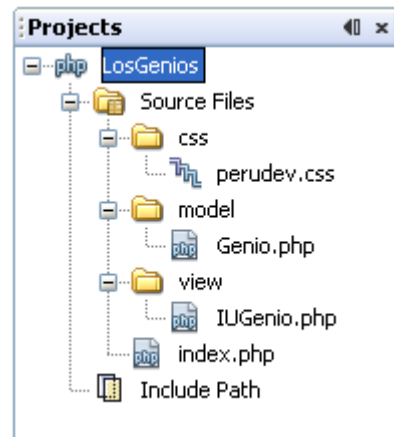
En este caso tenemos un programa PHP de nombre **IUGenio.php** que se encuentra en la carpeta **view**, este programa hará uso de la clase **Genio**, y gráficamente lo representamos de la siguiente manera:



#### 4.2.4 Estructura del Proyecto en NetBeans

Como se puede apreciar en el grafico de la derecha, tenemos tres carpetas, en la carpeta **css** tenemos la hoja de estilos, en la carpeta **model** ubicamos la clase **Genio**, y en la carpeta **view** el programa **IUGenio.php**.

En la raíz del proyecto tenemos el programa **index.php**, éste es el que inicia la ejecución del proyecto, y en este caso es hacer un redireccionamiento al programa **IUGenio.php**.



#### 4.2.5 Codificación de la Clase Genio

Archivo: Genio.php

```
<?php

class Genio {

    public function generarNumero(){
        $num = rand(1, 20);
        return $num;
    }

    public function generarOperador(){
        $num = rand(1,4);
        switch ($num) {
            case 1:
                $ope = "+";
                break;
            case 2:
                $ope = "-";
                break;
            case 3:
                $ope = "*";
                break;
            case 4:
                $ope = "/";
                break;
        }
    }
}
```

```
        return $ope;
    }

    public function procesar( $num1, $num2, $ope, $rptaUsua){
        switch ($ope) {
            case "+":
                $rptaSist = $num1 + $num2;
                break;
            case "-":
                $rptaSist = $num1 - $num2;
                break;
            case "*":
                $rptaSist = $num1 * $num2;
                break;
            case "/":
                $rptaSist = $num1 / $num2;
                break;
        }
        if( $rptaSist == $rptaUsua ){
            $msg = "Felicitaciones, respuesta correcta.";
        }else{
            $msg = "Respuesta incorrecta.";
        }
        $lista[] = array("Número 1",$num1);
        $lista[] = array("Número 2",$num2);
        $lista[] = array("Operador",$ope);
        $lista[] = array("Rpta. Sistema",$rptaSist);
        $lista[] = array("Rpta. Usuario",$rptaUsua);
        $lista[] = array("Mensaje",$msg);
        return $lista;
    }
}

?>
```

## 4.2.6 Codificación de la Interfaz de Usuario IUGenio

Archivo: IUGenio.php

```
<?php
require_once '../model/Genio.php';

$proceso = null;
if( isset( $_REQUEST["proceso"] ) ){
    $proceso = $_REQUEST["proceso"];
}

if( $proceso == "formulario" ) {
    $obj = new Genio();
    $num1 = $obj->generarNumero();
    $num2 = $obj->generarNumero();
    $ope = $obj->generarOperador();
}

if( $proceso == "resultado" ) {
    $num1 = $_REQUEST["num1"];
    $num2 = $_REQUEST["num2"];
    $ope = $_REQUEST["ope"];
    $rptaUsua = $_REQUEST["rpta"];
    $obj = new Genio();
    $lista = $obj->procesar($num1, $num2, $ope, $rptaUsua);
}
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <link rel="stylesheet" type="text/css" href="../css/perudev.css"/>
        <title>Untitled Document</title>
        <style type="text/css">
            <!--
            .style3 {
                font-size: 24px;
                text-align: center;
            }
            -->
        </style>
        <script>
```

```

function boton_generar(){
    document.location.href = "IUGenio.php?proceso=formulario";
}
</script>
</head>

<body>
    <h1>LOS GENIOS</h1>
    <p>
        <input name="generar" type="button" class="button" id="generar"
            value="Generar Operaci&oacute;n" onclick="boton_generar()"/>
    </p>

    <?php if($proceso == "formulario") { ?>
    <div id="formulario">
        <table width="298" border="1">
            <tr>
                <td width="92"><div align="center" class="style3">
                    <?php echo($num1) ?>
                </div></td>
                <td width="92"><div align="center" class="style3">
                    <?php echo($ope) ?>
                </div></td>
                <td width="92"><div align="center" class="style3">
                    <?php echo($num2) ?>
                </div></td>
            </tr>
        </table>
        <form id="form1" name="form1" method="post" action="IUGenio.php">
            <input type="hidden" name="proceso" value="resultado" />
            <input type="hidden" name="num1" value="<?php echo($num1) ?>" />
            <input type="hidden" name="num2" value="<?php echo($num2) ?>" />
            <input type="hidden" name="ope" value="<?php echo($ope) ?>" />
            <table width="300">
                <tr>
                    <td>Rpta.</td>
                    <td>
                        <input name="rpta" type="text" class="fieldEdit"
                            id="rpta" size="5" maxlength="5" />
                    </td>
                    <td>
                        <input name="procesar" type="submit" class="button"
                            id="procesar" value="Procesar" />
                    </td>
                </tr>
            </table>
        </form>
    </div>
    </?php>

```

```

</div>
<?php } ?>

<?php if( $proceso == "resultado" ) { ?>
<div id="resultado">
    <table width="435">
        <?php foreach( $lista as $rec ) { ?>
            <tr>
                <td width="113"><?php echo($rec[0]) ?></td>
                <td width="310"><?php echo($rec[1]) ?></td>
            </tr>
        <?php } ?>
    </table>
</div>
<?php } ?>
</body>
</html>

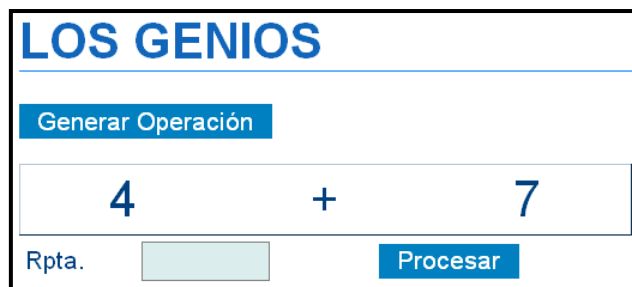
```

#### 4.2.7 Ejecución del Proyecto

Cuando ejecutamos el proyecto inicialmente tenemos la siguiente interfaz:



Al hacer clic en el botón **Generar Operación** obtenemos una operación en forma aleatoria, tal como se ilustra a continuación:



En esta nueva interfaz el usuario tiene dos alternativas:

1. Generar una nueva operación.
2. Ingresar la respuesta de la operación planteada y la procesa haciendo clic en el botón **Procesar**.

Supongamos que eligió la segunda opción, el resultado se muestra de la siguiente manera:

## LOS GENIOS

Generar Operación

Número 1	4
Número 2	7
Operador	+
Rpta. Sistema	11
Rpta. Usuario	15
Mensaje	Respuesta incorrecta.

El mensaje depende del resultado de comparar la respuesta del sistema con la respuesta del usuario, para este caso no son iguales por lo tanto el mensaje refleja tal situación.



## 5 Proyectos Propuestos

---

### 5.1 Encontrar el Promedio de un Alumno

El administrador de cursos de la institución educativa **EduTech** necesita un programa para calcular el promedio de un alumno, se debe tomar en cuenta lo siguiente:

3. Son cuatro notas de práctica, de donde se obtiene un promedio de prácticas (PP), se promedian las tres mejores notas.
4. Se tiene también un examen parcial (EP) y un examen final (EF).
5. El promedio final (PF) se obtiene aplicando la siguiente fórmula:

$$PF = PP * 0.30 + EP * 0.30 + EF * 0.40$$

Se pide plantear la solución e implementarla aplicando los conceptos de POO.

### 5.2 Encontrar el Importe de una Venta

El dueño de una tienda necesita un programa para encontrar el importe de una venta, se debe tomar en cuenta lo siguiente:

6. El precio de venta ya incluye el IGV.
7. Cada venta es un solo tipo de artículo.
8. El cliente puede comprar varias unidades del artículo.
9. En función a la cantidad de unidades que el cliente está comprando, el vendedor puede hacerle un descuento sobre el precio de venta que resulta de una negociación en el acto de la venta con el cliente, este descuento en ningún caso puede ser superior al 10% del precio de venta.

Se pide plantear la solución e implementarla aplicando los conceptos de POO.