Control semana 3: Programación Orientada a Objetos con PHP

Jorge Peñaloza

Programación Avanzada I

Instituto IACC

28 de junio de 2021

**Desarrollo**

**Respuestas del problema**

***Contexto***

Basándose en el diseño lógico realizado en el control de la semana 2, en el cual decía que:

Usted ha sido seleccionado para diseñar lógicamente (puede usar elementos gráficos como formas) un sistema para el registro de estudiantes de colegios privados de la ciudad de Santiago, este sistema debe manejar las personas dentro del proceso como son: estudiantes con todos sus atributos ( nombre, apellido, rut, dirección) y apoderados también con sus atributos ( nombre, apellido, rut, dirección), que año y asignaturas cursan cada uno de los estudiantes, si realizan actividades extras, especificando cual actividad realizan.

Ahora bien, basándose en esa situación ejemplificar gráficamente:

1. Clases abstractas, rasgos, clases anónimas y sobrecarga.
2. Muestre como seria la iteración de objetos con los métodos mágicos y palabras clave final.

Recuerde que debe mostrar todo gráficamente y justificar su elaboración.

***Diseño de prototipo***

La descripción de cada archivo del proyecto se puede observar en la tabla 1 la cual está compuesta por archivos con extensión php.

En el diagrama UML de clases se puede observar en la figura 1, en este diagrama se puede apreciar la relación que existe entre las clases, clases abstractas y trait.

Tabla 1

*Archivos del proyecto*

|  |  |
| --- | --- |
| Nombre archivo | Descripción |
| index.php | Contiene el cuerpo principal con un ejemplo de cómo utilizar las clases propuestas. Contiene un auto cargador de clases. Además, contiene el ejemplo de como crear una clase anónima y uso de trait. |
| Persona.php | Este archivo contiene la clase padre *Persona* con uso de trait dentro de una clase que hereda propiedades de una clase abstracta. |
| PersonaA.php | Esta clase contiene el ejemplo de como implementar una clase abstracta, la cual luego hereda sus propiedades a la clase Persona. |
| Apoderado.php | Este archivo contiene la clase hija *Apoderado* que hereda sus propiedades básicas de la clase *Persona* |
| Estudiante.php | Este archivo contiene la clase hija *Estudiante* que hereda sus propiedades básicas de la clase *Persona* |
| Asignatura.php | Este archivo contiene la clase Asignatura que contiene la implementación de métodos mágicos. |
| Actividad.php | Este archivo contiene la clase Actividad que parte de implementar métodos mágicos con constructor y destructor, también se implementan métodos de sobrecarga para agregar una propiedad con su respectivo contenido y visualizar el contenido de esa propiedad con el método con sobrecarga \_\_get(). Además implementa la visulizacion de las propiedades de una clase con un método que permite la iteración de objetos. |

Fuente: Elaboración propia

Figura 1

*Diagrama UML de clases de PHP del proyecto*

En la *figura 2* se pueden observar los atributos y métodos de la clase Persona con sus respectivos atributos públicos y un método para validar RUT obtenido de (Barriga, s.f.), el cual se utilizó para verificar la funcionalidad de método en la herencia de clases.

Diagrama

Descripción generada automáticamente

Fuente: Elaboración propia

En la *figura 2* se pueden observar los atributos y métodos de la clase abstracta *PersonaA* con sus respectivos métodos abstractos. Se notar que en los métodos abstractos dentro de la abstracta solo se define la existencia del método, pero no su implementación.

Además, se puede ver su código fuente en Código fuente 1 para confirmar su implementación.

Figura 2

*Diagrama UML de clase abstracta PersonaA*

Imagen que contiene Diagrama

Descripción generada automáticamente

Fuente: Elaboración propia

Código fuente 1

*Archivo: PersonaA.php*

<?php

// Ejemplo de implementación de clase abstracta

abstract class PersonaA

{

    abstract protected function rutIsValid( $rutA);

    abstract protected function salir();

}

?>

Fuente: Elaboración propia

En la *figura 3* se pueden la implementación del trait *Salir*, y en la figura 4 se observa la clase *Persona*, se debe notar que la clase P*ersona* hereda 2 metodos de la clase abstracta *PersonaA* y utiliza el trait *Salir* en su implementación.Una característica relevante de la clase persona es la implementación del método abstracto heredado de *PersonaA* el cual permite utilizarlo como público mediante el uso de otro método que realiza llamadas internas a el además mejora su funcionalidad en caso de ser necesario realizar validación de rut en caso de ser necesario*.*

La implementación de el *trait* Salir y la clase *Persona* se puede ver en *Código fuente 2.*

Figura 3

*Diagrama UML de trait Salir*

Diagrama

Descripción generada automáticamente con confianza media

Fuente: Elaboración propia

Figura 4

*Diagrama UML de clase Persona*

Diagrama

Descripción generada automáticamente

Fuente: Elaboración propia

Código fuente 2

*Archivo: Persona.php*

<?php

trait Salir

{

    public function salir()

    {

        echo "<br>Desctructor de clases se ha activado<br>";

    }

}

class Persona extends PersonaA

{

    use Salir;

    public $nomsbre;

    public $apellido;

    public $rut;

    public $direccion;

    // Ejemplos de metodos magicos

    public function \_\_construct( $nombre, $apellido, $rut, $direccion )

    {

        $this->nombre = $nombre;

        $this->apellido = $apellido;

        $this->rut = $rut;

        $this->direccion = $direccion;

    }

    public function \_\_destruct()

    {

        $this->salir();

    }

    //La validacion de rut se ha obtenido de https://gist.github.com/rbarrigav/3881019#file-validar\_rut-php

    protected function rutIsValid( $rutA)

    {

        $rut = $rutA;

        $rut = preg\_replace('/[^k0-9]/i', '', $rut);

        $dv  = substr($rut, -1);

        $numero = substr($rut, 0, strlen($rut)-1);

        $i = 2;

        $suma = 0;

        foreach(array\_reverse(str\_split($numero)) as $v)

        {

            if($i==8)

                $i = 2;

            $suma += $v \* $i;

            ++$i;

        }

        $dvr = 11 - ($suma % 11);

        if($dvr == 11)

            $dvr = 0;

        if($dvr == 10)

            $dvr = 'K';

        if($dvr == strtoupper($dv))

            return true;

        else

            return false;

    }

    public function rutEsvalido($rutA = Null)

    {

        return $this->rutIsValid(isset($rutA) ? $rutA : $this->rut);

    }

    public function \_\_isset($name)

    {

        echo "¿Está definido '$name'?\n";

        return isset($this->data[$name]);

    }

}

?>

Fuente: Elaboración propia

En la *figura 5* se pueden observar los atributos y métodos de la clase *Apoderado* con su respectivo atributo privado y métodos públicos insertar, eliminar y obtener elementos en el atributo privado.

En esta clase se heredan los atributos de la clase *Persona* y se implementan los atributos y métodos propios a esta clase. En *Código fuente 3* se puede apreciar esta implementación.

Figura 5

*Diagrama UML de clase Apoderado*

Tabla

Descripción generada automáticamente

Fuente: Elaboración propia

Código fuente 3

*Archivo: Apoderado.php*

<?php

class Apoderado extends Persona

{

    private $estudiantes = array();

    public function addEstudiantes( $rut )

    {

        $this->estudiantes[] = $rut;

    }

    public function delEstudiantes( $rut )

    {

        $this->estudiantes = array\_diff($this->estudiantes, array($rut));

    }

    public function obtenerEstudiantes()

    {

        return $this->estudiantes;

    }

}

?>

Fuente: Elaboración propia

En la *figura 6* se pueden observar los atributos y métodos de la clase y su implementación se ve *Código fuente 4*, se debe tomar en cuenta que los métodos solo se implementaron como ejemplo, ya que no tienen funcionalidad.

Figura 6

*Diagrama UML de clase Estudiante*

Imagen que contiene Tabla

Descripción generada automáticamente

Fuente: Elaboración propia

Código fuente 4

*Archivo: Estudiante.php*

<?php

class Estudiante extends Persona

{

    public $anioCurso;

    public function \_\_construct( $nombre, $apellido, $rut, $direccion, $anioCurso )

    {

        $this->nombre = $nombre;

        $this->apellido = $apellido;

        $this->rut = $rut;

        $this->direccion = $direccion;

        $this->anioCurso = $anioCurso;

    }

    public function obtenerAsignaturas( )

    {

        return "Obtener Asignaturas";

    }

    public function obtenerActividades( )

    {

        return "Obtener Actividades";

    }

    public function obtenerApoderado( )

    {

        return "Obtener Apoderado";

    }

}

?>

Fuente: Elaboración propia

En la *figura 7* se pueden observar los atributos y métodos de la clase *Asignatura* con sus respectivos atributos y métodos para agregar, eliminar y obtener datos del atributo *estudiantes*.

En *Código fuente 5* se puede apreciar esta implementación. En esta clase se implementaron métodos constructor y destructor ya que es una clase padre.

Figura 7

*Diagrama UML de clase Asignatura*

Tabla

Descripción generada automáticamente

Fuente: Elaboración propia

Código fuente 5

*Archivo: Asignatura.php*

<?php

class Asignatura

{

    use Salir;

    public $codigo;

    public $descripcion;

    public $planEstudios;

    private $estudiantes = array();

    public function \_\_construct( $codigo, $descripcion, $planEstudios )

    {

        $this->codigo = $codigo;

        $this->descripcion = $descripcion;

        $this->planEstudios = $planEstudios;

    }

    public function \_\_destruct()

    {

        $this->salir();

    }

    public function addEstudiantes( $rut )

    {

        $this->estudiantes[] = $rut;

    }

    public function delEstudiantes( $rut )

    {

        $this->estudiantes = array\_diff($this->estudiantes, array($rut));

    }

    public function obtenerEstudiantes()

    {

        return $this->estudiantes;

    }

}

?>

Fuente: Elaboración propia

En la *figura 8* se pueden observar los atributos y métodos de la clase *Actividad* con sus respectivos atributos y métodos para agregar, eliminar y obtener datos del atributo *estudiantes*. Se debe considerar que esta clase contiene el atributo *data* que es un array que contendrá la información de cada atributo y su respectivo valor, de tal forma que ésta clase permite modificar sus atributos en forma dinámica debido al uso de los métodos mágicos *\_\_get()* y *\_\_set()* que debido a la sobrecarga se produce al utilizar un atributo no existente, ya que éste, en dinámica utiliza el array *data* para crear el atributo en ese momento.

Otra característica de esta clase es que se implemento en ella un método publico que listar los atributos del objeto en tiempo de ejecución. Ésta acción se realiza utilizando la iteración de objetitos,

En *Código fuente 6* se puede apreciar esta implementación.

Figura 8

*Diagrama UML de clase Actividad*

Tabla

Descripción generada automáticamente con confianza baja

Fuente: Elaboración propia

Código fuente 6

*Archivo: Actividad.php*

<?php

class Actividad

{

    protected $data = array();

    private $estudiantes = array();

    public function \_\_construct( $codigo, $descripcion )

    {

        $this->codigo = $codigo;

        $this->descripcion = $descripcion;

    }

    public function \_\_destruct()

    {

        $this->salir();

    }

    public function addEstudiantes( $rut )

    {

        $this->estudiantes[] = $rut;

    }

    public function delEstudiantes( $rut )

    {

        $this->estudiantes = array\_diff($this->estudiantes, array($rut));

    }

    public function obtenerEstudiantes()

    {

        return $this->estudiantes;

    }

    private function salir()

    {

        echo "<br>Desctructor de clases se ha activado<br>";

    }

    public function \_\_set($name, $value)

    {

        echo "Estableciendo ".$name." a ".$value."<br>";

        $this->data[$name] = $value;

    }

    public function \_\_get($name)

    {

        echo "Consultando '$name'\n";

        if (array\_key\_exists($name, $this->data)) {

            return $this->data[$name];

        }

        $trace = debug\_backtrace();

        trigger\_error(

            ' <strong>Mesaje de comprobacion creado para probar sobrecarga y metodos magicos </strong><br>'.

            ' Propiedad indefinida mediante \_\_get(): ' . $name .

            ' en ' . $trace[0]['file'] .

            ' en la línea ' . $trace[0]['line'],

            E\_USER\_NOTICE);

        return null;

    }

    // Esta funcion obtenida de php.net se modifico para que cuando el tipo de dato sea un array, pueda recorrerlo por

    function iterateVisible()

    {

        foreach ($this as $clave => $valor)

        {

            echo "<br> tipo de dato: ".gettype($valor)."<br>";

            if(gettype($valor)=="array")

            {

                $i = 0;

                foreach ($valor as $key => $value)

                {

                    echo $i."-".$key." => ".$value."<br>";

                    $i++;

                }

            }

            else

            {

                print $clave." => ".$valor."\n";

            }

        }

     }

}

?>

Fuente: Elaboración propia

En *Código fuente 7* se puede observar el código fuente de un ejemplo de la implementación y en la figura 9 se observa cómo se visualiza el ejemplo planteado.

Código fuente 7

*Archivo: index.php*

<?php

spl\_autoload\_register( function ($nombre\_clase)

{

    include $nombre\_clase . '.php';

});

// Ejemplo de uso de clase anonima

echo "Prueba clase \*Apoderado <br>";

$apoderado  = new Class('Segundo','Peñaloza','13834570-K','Maria Elena') extends Persona

{

    private $estudiantes = array();

    public function addEstudiantes( $rut )

    {

        $this->estudiantes[] = $rut;

    }

    public function delEstudiantes( $rut )

    {

        $this->estudiantes = array\_diff($this->estudiantes, array($rut));

    }

    public function obtenerEstudiantes()

    {

        return $this->estudiantes;

    }

};

echo $apoderado->nombre."<br>";

echo $apoderado->apellido."<br>";

echo $apoderado->rut."<br>";

echo $apoderado->direccion."<br>";

echo ($apoderado->rutEsvalido()?"Verdadero":"Falso")."<br>";

$apoderado->addEstudiantes("1383470-K");

$apoderado->addEstudiantes("111111111-1");

print\_r($apoderado->obtenerEstudiantes());

echo "<br>";

$apoderado->delEstudiantes("111111111-1");

print\_r($apoderado->obtenerEstudiantes());

echo "<br>";

echo "<br>Prueba clase \*Estudiante <br>";

$estudiante  = new Estudiante('Jorge','Peñaloza','13834570-k','Maria Elena', 2021);

echo $estudiante->nombre."<br>";

echo $estudiante->apellido."<br>";

echo $estudiante->rut."<br>";

echo $estudiante->direccion."<br>";

echo $estudiante->anioCurso."<br>";

echo ($estudiante->rutEsvalido()?"Verdadero":"Falso")."<br>";

echo ($estudiante->obtenerAsignaturas())."<br>";

echo ($estudiante->obtenerActividades())."<br>";

echo ($estudiante->obtenerApoderado())."<br>";

echo "<br>";

echo "Prueba clase \*Asignatura <br>";

$asignatura  = new Asignatura( 22,'Programacion Avanzada I','Plan de estudios');

echo $asignatura->codigo."<br>";

echo $asignatura->descripcion."<br>";

echo $asignatura->planEstudios."<br>";

$asignatura->addEstudiantes("13834570-K");

$asignatura->addEstudiantes("111111111-1");

print\_r($asignatura->obtenerEstudiantes());

echo "<br>";

$asignatura->delEstudiantes("111111111-1");

print\_r($asignatura->obtenerEstudiantes());

echo "<br>";

echo "<br>";

echo "Prueba clase \*Actividad <br>";

$actividad  = new Actividad( 25,'Golf');

echo "<br>";

echo $actividad->codigo."<br>";

echo $actividad->descripcion."<br>";

echo $actividad->descripcion\_test."<br>";

$actividad->addEstudiantes("13834570-K");

$actividad->addEstudiantes("2-2");

print\_r($actividad->obtenerEstudiantes());

echo "<br>";

$actividad->delEstudiantes("13834570-K");

print\_r($actividad->obtenerEstudiantes());

echo "<br>";

echo "<br><strong>Prueba de iteracion en clase con metodos magicos y palabra clave final</strong><br>";

$actividad->addEstudiantes("13834570-K");

$actividad->addEstudiantes("3-3");

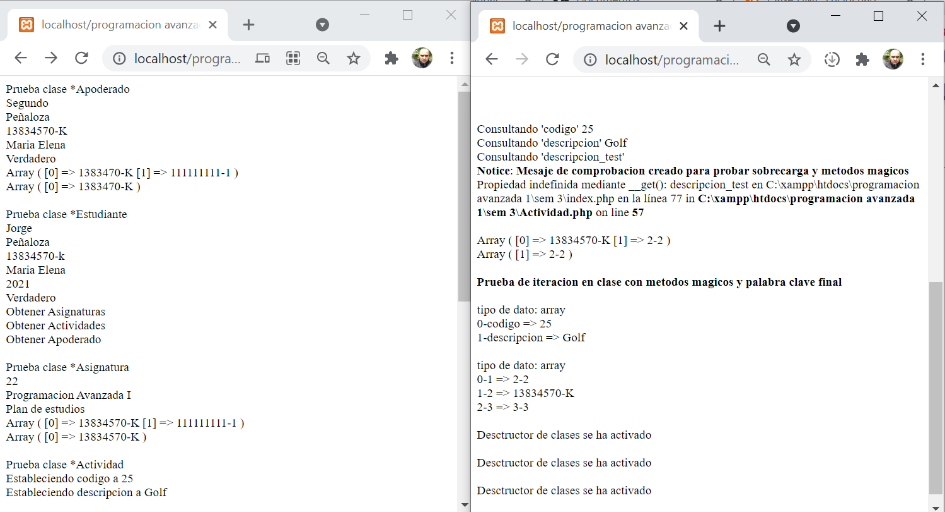
$actividad->iterateVisible()

?>

Fuente: Elaboración propia

Figura 9

*Vista en navegador web de Código fuente 7*



Fuente: Elaboración propia

Para ver la implementación de la clase se hace referencia a Código fuente 8 que contiene un fragmento de index.php en que se ve como implementar una clase anónima, y además en la figura 10 se observa cómo se visualizan los datos de ese fragmento de código fuente.

Código fuente 8

*Ejemplo de implementación de clase anónima y ejemplo de uso*

// Ejemplo de uso de clase anonima

echo "Prueba clase \*Apoderado <br>";

$apoderado  = new Class('Segundo','Peñaloza','13834570-K','Maria Elena') extends Persona

{

    private $estudiantes = array();

    public function addEstudiantes( $rut )

    {

        $this->estudiantes[] = $rut;

    }

    public function delEstudiantes( $rut )

    {

        $this->estudiantes = array\_diff($this->estudiantes, array($rut));

    }

    public function obtenerEstudiantes()

    {

        return $this->estudiantes;

    }

};

echo $apoderado->nombre."<br>";

echo $apoderado->apellido."<br>";

echo $apoderado->rut."<br>";

echo $apoderado->direccion."<br>";

echo ($apoderado->rutEsvalido()?"Verdadero":"Falso")."<br>";

$apoderado->addEstudiantes("1383470-K");

$apoderado->addEstudiantes("111111111-1");

print\_r($apoderado->obtenerEstudiantes());

echo "<br>";

$apoderado->delEstudiantes("111111111-1");

print\_r($apoderado->obtenerEstudiantes());

echo "<br>";

Fuente: Elaboración propia

Figura 10

*Vista de ejemplo de clase anónima, en este caso de Apoderado*

Texto, Carta

Descripción generada automáticamente

Fuente: Elaboración propia

Para ver la implementación de la clase se hace referencia a Código fuente 9 que contiene un fragmento de index.php se puede ver cómo utilizar la clase *Estudiante*, y además en la figura 11 se observa cómo se visualizan los datos de ese fragmento de código fuente.

Código fuente 9

*Ejemplo de visualización de clase Estudiante*

echo "<br>Prueba clase \*Estudiante <br>";

$estudiante  = new Estudiante('Jorge','Peñaloza','13834570-k','Maria Elena', 2021);

echo $estudiante->nombre."<br>";

echo $estudiante->apellido."<br>";

echo $estudiante->rut."<br>";

echo $estudiante->direccion."<br>";

echo $estudiante->anioCurso."<br>";

echo ($estudiante->rutEsvalido()?"Verdadero":"Falso")."<br>";

echo ($estudiante->obtenerAsignaturas())."<br>";

echo ($estudiante->obtenerActividades())."<br>";

echo ($estudiante->obtenerApoderado())."<br>";

echo "<br>";

Fuente: Elaboración propia

Figura 11

*Vista de ejemplo de uso de clase Estudiante*

Texto

Descripción generada automáticamente

Fuente: Elaboración propia

Para ver la implementación de la clase se hace referencia a Código fuente 10 que contiene un fragmento de index.php se puede ver cómo utilizar la clase *Asignatura*, y además en la figura 12 se observa cómo se visualizan los datos de ese fragmento de código fuente.

Código fuente 10

*Ejemplo de visualización de clase Asignatura*

echo "Prueba clase \*Asignatura <br>";

$asignatura  = new Asignatura( 22,'Programacion Avanzada I','Plan de estudios');

echo $asignatura->codigo."<br>";

echo $asignatura->descripcion."<br>";

echo $asignatura->planEstudios."<br>";

$asignatura->addEstudiantes("13834570-K");

$asignatura->addEstudiantes("111111111-1");

print\_r($asignatura->obtenerEstudiantes());

echo "<br>";

$asignatura->delEstudiantes("111111111-1");

print\_r($asignatura->obtenerEstudiantes());

Fuente: Elaboración propia

Figura 12

*Vista de ejemplo de uso de clase Asignatura*

Texto

Descripción generada automáticamente

Fuente: Elaboración propia

Para ver la implementación de la clase se hace referencia a Código fuente 11 que contiene un fragmento de index.php se puede ver cómo utilizar la clase *Actividad*, y además en la figura 13 se observa cómo se visualizan los datos de ese fragmento de código fuente.

Código fuente 11

*Ejemplo de visualización de clase Actividad*

echo "Prueba clase \*Actividad <br>";

$actividad  = new Actividad( 25,'Golf');

echo "<br>";

echo $actividad->codigo."<br>";

echo $actividad->descripcion."<br>";

echo $actividad->descripcion\_test."<br>";

$actividad->addEstudiantes("13834570-K");

$actividad->addEstudiantes("2-2");

print\_r($actividad->obtenerEstudiantes());

echo "<br>";

$actividad->delEstudiantes("13834570-K");

print\_r($actividad->obtenerEstudiantes());

echo "<br>";

echo "<br><strong>Prueba de iteracion en clase con metodos magicos y palabra clave final</strong><br>";

$actividad->addEstudiantes("13834570-K");

$actividad->addEstudiantes("3-3");

$actividad->iterateVisible()

Fuente: Elaboración propia

Figura 13

*Vista de ejemplo de uso de clase Actividad*

Texto

Descripción generada automáticamente

Fuente: Elaboración propia

El código fuente del proyecto se encuentra en el archivo Jorge\_Peñaloza\_Control3.rar adjunto.

Bibliografía

Barriga, R. (s.f.). *GitHub*. Obtenido de https://gist.github.com/rbarrigav/3881019#file-validar\_rut-php

IACC. (2020). Programación Orientada a Objetos con PHP. En *Programación avanzada 1. Semana 3.*

IACC. (2021). Tarea semana 3.

The PHP Group. (s.f.). *Manual de PHP*. Obtenido de https://www.php.net/manual/es/index.php

*w3schools.com*. (s.f.). Obtenido de https://www.w3schools.com/