Tipos de Datos abstractos

Algortimos TUPAR

Problemática

Se requiere un programa que maneje las posibles operaciones de un número natural. Entre ellas:

- Definir un numero natural.
- Setearlo en un valor dado.
- Devolver siguiente.
- Incrementarlo en un valor dado.
- Decrementarlo en un valor dado.
- Resetear (volverlo a 0).

Posible solución

- Definir funciones para realizar todas las operaciones requeridas e ir pasando variables para modificarlas.

Posible solución

```php
<?php

function setValue(&$numero, $valor){
    //...//
}

function resetear(&$numero){
    //...//
}

function increment(&$numero){
    //...//
}

function decrement(&$numero){
    //...//
}

function nextNatural($numero){
    //...//
}

?>
```

naturales.php

```php
<?php

require_once("naturales.php");

$natural=0;

/*
    llamar a las funciones requeridas
*/

?>
```

Posible solución

- Se usa un entero externo a las funciones.
- No hay encapsulamiento.
- Cualquier puede acceder a las funciones.

```php
<?php

function setValue(&$numero, $valor){
    //...//
}

function resetear(&$numero){
    //...//
}

function increment(&$numero){
    //...//
}

function decrement(&$numero){
    //...//
}

function nextNatural($numero){
    //...//
}

?>
```

naturales.php

```php
<?php

require_once("naturales.php");

$natural=0;

/*
    llamar a las funciones requeridas
*/

?>
```

Tipo de dato abstracto

Una mejor solución consiste en definir un tipo de dato que tenga el comportamiento de un numero natural.

Las funciones son parte de la instancia del natural.

```php
$n=new Natural();    //defino un tipo de dato natural

$n->setValue(3);     //lo seteo en 3
$n->increment(8);    //incremento en 8
echo $n->nextNatural(); //imprimo el siguiente
```

Tipo de dato abstracto

Una mejor solución consiste en definir un tipo de dato que tenga el comportamiento de un numero natural.

Las funciones son parte de la instancia del natural.

Defino un tipo de dato NO PRIMITIVO

```
$n=new Natural();    //defino un tipo de dato natural

$n->setValue(3);     //lo seteo en 3
$n->increment(8);    //incremento en 8
echo $n->nextNatural(); //imprimo el siguiente
```

Tipo de dato abstracto

Una mejor solución consiste en definir un tipo de dato que tenga el comportamiento de un numero natural.

Las funciones son parte de la instancia del natural.

Defino un tipo de dato NO PRIMITIVO

Estructura??

```
$n=new Natural();    //defino un tipo de dato natural

$n->setValue(3);     //lo seteo en 3
$n->increment(8);    //incremento en 8
echo $n->nextNatural(); //imprimo el siguiente
```

Tipo de dato abstracto

Una mejor solución consiste en definir un tipo de dato que tenga el comportamiento de un numero natural.

Las funciones son parte de la instancia del natural.

Defino un tipo de dato NO PRIMITIVO

Estructura??

```
$n=new Natural();    //defino un tipo de dato natural

$n->setValue(3);     //lo seteo en 3
$n->increment(8);    //incremento en 8
echo $n->nextNatural(); //imprimo el siguiente
```

Hay funciones asociadas a la variable de tipo “Natural”

Clases

Clases en PHP: Tipos de datos definidos por el usuario que almacenan atributos y métodos.

Estos atributos y métodos son accesibles o no según su privacidad de acceso (public, private).

La inicialización se da por el constructor de la clase

Clases

Una clase va a representar algo abstracto para poder programar de manera más legible.

Ej:

- Una persona
- Un auto
- Un grafo
- Un archivo
- Una lista
- ...

Clase vs Instancia

Una vez definida la clase, puede usarse para definirle un tipo a una variable. Esta variable, será una instancia de la clase definida.

```
class Persona
{

    /*
        ...
    */

}
```

```
$p1=new Persona();
```

Clases

Dentro de la clase se definirán variables (atributos) y funciones (métodos) que manipulan dichas variables.

Cada una de estas variables o funciones tienen un nivel de acceso asociado:

public: accesibles desde dentro y fuera de la clase.

private: accesibles sólo desde dentro de la clase.

Clases

Ej: Clase persona.
Una persona tiene:

- Nombre.
- Apellido.
- DNI.
- Edad.

Se requiere:

- Nombre completo
- DNI
- Diferencia de edad con otra persona.
- Incrementar su edad.

Clases

Ej: Clase persona.
Una persona tiene:

- Nombre.
- Apellido.
- DNI.
- Edad.

Atributos

Se requiere:

- Nombre completo
- DNI
- Diferencia de edad con otra persona.
- Incrementar su edad.

Métodos

Clases

```php
class Persona
{
    private $nombre;
    private $apellido;
    private $dni;
    private $edad;


    public function __construct($nombre, $apellido, $dni, $edad){
        $this->nombre=$nombre;
        $this->apellido=$apellido;
        $this->dni=$dni;
        $this->edad=$edad;
    }

    public function nombreCompleto(){
        return $this->nombre . " " . $this->apellido;
    }

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

    public function devolverEdad(){
        return $this->edad;
    }

    public function diferenciaEdad(Persona $p1){
        return $this->edad - $p1->devolverEdad();
    }

    public function cumplirAnios(){
        $this->edad++;
    }

}
```

Clases

Atributos

Métodos

```php
class Persona
{
    private $nombre;
    private $apellido;
    private $dni;
    private $edad;


    public function __construct($nombre, $apellido, $dni, $edad){
        $this->nombre=$nombre;
        $this->apellido=$apellido;
        $this->dni=$dni;
        $this->edad=$edad;
    }

    public function nombreCompleto(){
        return $this->nombre . " " . $this->apellido;
    }

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

    public function devolverEdad(){
        return $this->edad;
    }

    public function diferenciaEdad(Persona $p1){
        return $this->edad - $p1->devolverEdad();
    }

    public function cumplirAnios(){
        $this->edad++;
    }

}
```

Clases

Atributos

No accesible desde fuera de la clase

Métodos

```php
class Persona
{
    private $nombre;
    private $apellido;
    private $dni;
    private $edad;

    public function __construct($nombre, $apellido, $dni, $edad){
        $this->nombre=$nombre;
        $this->apellido=$apellido;
        $this->dni=$dni;
        $this->edad=$edad;
    }

    public function nombreCompleto(){
        return $this->nombre . " " . $this->apellido;
    }

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

    public function devolverEdad(){
        return $this->edad;
    }

    public function diferenciaEdad(Persona $p1){
        return $this->edad - $p1->devolverEdad();
    }

    public function cumplirAnios(){
        $this->edad++;
    }

}
```

Clases

Atributos

No accesible desde fuera de la clase

Métodos

Accesible desde fuera de la clase

```php
class Persona
{
    private $nombre;
    private $apellido;
    private $dni;
    private $edad;

    public function __construct($nombre, $apellido, $dni, $edad){
        $this->nombre=$nombre;
        $this->apellido=$apellido;
        $this->dni=$dni;
        $this->edad=$edad;
    }

    public function nombreCompleto(){
        return $this->nombre . " " . $this->apellido;
    }

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

    public function devolverEdad(){
        return $this->edad;
    }

    public function diferenciaEdad(Persona $p1){
        return $this->edad - $p1->devolverEdad();
    }

    public function cumplirAnios(){
        $this->edad++;
    }
}
```

Clases

Constructor: Método que se llama al crear una instancia de la clase

```php
class Persona
{
    private $nombre;
    private $apellido;
    private $dni;
    private $edad;

    public function __construct($nombre, $apellido, $dni, $edad){
        $this->nombre=$nombre;
        $this->apellido=$apellido;
        $this->dni=$dni;
        $this->edad=$edad;
    }

    public function nombreCompleto(){
        return $this->nombre . " " . $this->apellido;
    }

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

    public function devolverEdad(){
        return $this->edad;
    }

    public function diferenciaEdad(Persona $p1){
        return $this->edad - $p1->devolverEdad();
    }

    public function cumplirAnios(){
        $this->edad++;
    }

}
```

Clases

Para acceder a un atributo desde dentro de la clase, usar $this

```php
class Persona
{
    private $nombre;
    private $apellido;
    private $dni;
    private $edad;


    public function __construct($nombre, $apellido, $dni, $edad){
        $this->nombre=$nombre;
        $this->apellido=$apellido;
        $this->dni=$dni;
        $this->edad=$edad;
    }

    public function nombreCompleto(){
        return $this->nombre . " " . $this->apellido;
    }

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

    public function devolverEdad(){
        return $this->edad;
    }

    public function diferenciaEdad(Persona $p1){
        return $this->edad - $p1->devolverEdad();
    }

    public function cumplirAnios(){
        $this->edad++;
    }

}
```

Clases

```php
$p=new Persona("Juan", "Perez", "34433344", 20);
$p1=new Persona("Javier", "Gonzalez", "36633366", 17);


echo $p->diferenciaEdad($p1);
```

Imprime 3

Clases

Buenas prácticas:

- Hacer bosquejo de atributos y métodos de una clase antes de programarla.
- Definir los atributos como privados. El accesos a ellos se debe hacer mediantes metodos (get), asi como también la modificación (set).
- Reusar clases! Ej: definir la clase lista y la clase pila usando una lista