CONCEPTOS FUNDAMENTALES DE YUI 2

Ricardo Pérez López

IES Doñana, curso 2017-18

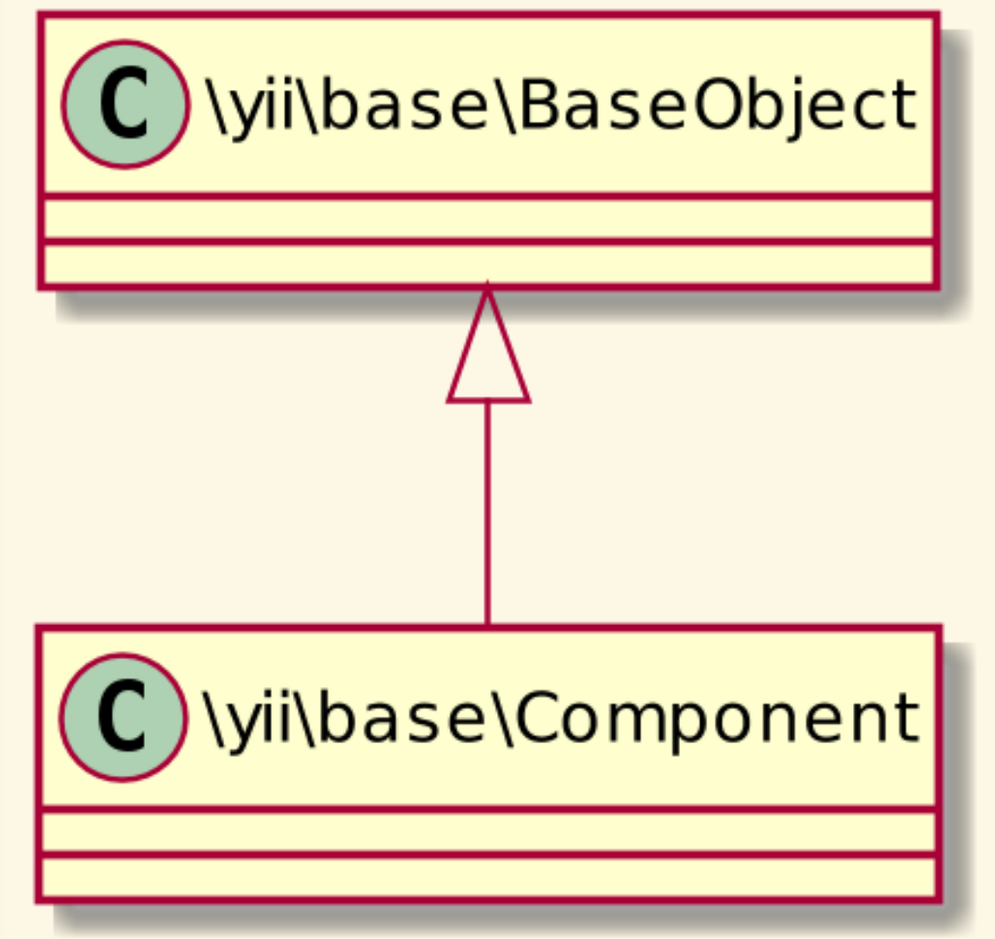
COMPONENTES

COMPONENTES

Se caracterizan por tener:

- Propiedades
- Configurabilidad
- Eventos
- Comportamientos (*behaviors*)

Las dos primeras características se heredan de `\yii\base\BaseObject`.


\yii\base\BaseObject
\yii\base\Component

\yii\base\BaseObject

Introduce las siguientes características:

- Propiedades
- Configurabilidad

PROPIEDADES

PROPIEDADES

- En PHP, a las variables miembro de una clase (variables de instancia) se las denomina también *propiedades*.
- Esas variables son parte de la definición de la clase, y se usan para representar el estado de una instancia de dicha clase.
- La clase `\yii\base\BaseObject` de Yii 2 permite crear propiedades a partir de métodos *getter* y *setter*.
- Toda clase que herede (directa o indirectamente) de `\yii\base\BaseObject` podrá definir propiedades de esa manera.

Por ejemplo:

```php
namespace app\components;

class Foo extends \yii\base\BaseObject
{
    private $_label;

    // El método getter:
    public function getLabel()
    {
        return $this-> label;
    }

    // El método setter:
    public function setLabel($value)
    {
        $this->_label = trim($value);
    }
}
```

Crea la propiedad `label`, accesible mediante `$foo->label`.

```php
$foo = new Foo;

echo $foo->label;      // Llama internamente a $foo->getLabel()

$foo->label = 'hola'; // Llama internamente a $foo->setLabel('hola');
```

Como `setLabel($value)` está definido como:

```php
public function setLabel($value)
{
    $this->_label = trim($value);
}
```

al asignarle una cadena a la propiedad se *trimeará* automáticamente, eliminando los espacios sobrantes:

```php
$foo->label = '    hola    ';   // Se guarda sin espacios sobrantes

echo $foo->label;               // Devuelve "hola" (sin espacios)
```

PROPIEDADES DE SÓLO LECTURA

Si definimos sólo el *getter* y no el *setter*, crearemos una **propiedad de sólo lectura**, por lo que podremos consultar su valor pero no cambiarlo:

```php
class Prueba extends \yii\base\BaseObject
{
    public $_valor = 25;

    // Método getter (no hay setter):
    public function getValor()
    {
        return $this->_valor;
    }
}

$p = new Prueba;
echo $p->valor;  // Devuelve 25
$p->valor = 30;  // Da ERROR
```

CONFIGURABILIDAD

CONFIGURABILIDAD

- Una instancia de la clase `\yii\base\BaseObject` (o de una subclase suya) permite ser *configurado*.
- Una **configuración** es simplemente un array que contiene parejas de `clave => valor`, donde la `clave` representa el nombre de una propiedad (una cadena), y el `valor` es el valor que queremos asignarle a dicha propiedad.
- Se pueden usar para:
  - Asignar valores de forma masiva a las propiedades de un objeto usando `Yii::configure($objeto, $config)`.
  - Crear una instancia asignándole valores iniciales a sus propiedades usando `Yii::createObject($config)`.
  - Más posibilidades que iremos viendo en su momento.

ASIGNACIÓN MASIVA

Supongamos la siguiente clase:

```php
use \yii\base\BaseObject;

class Prueba extends BaseObject
{
    public $uno;
    private $_dos;

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

    public function setDos($dos)
    {
        $this->_dos = $dos;
    }
}
```

Algunas posibles configuraciones:

```php
[ 'uno' => 5, 'dos' => 7 ]

[ 'dos' => 18 ]
```

Se pueden aplicar a un objeto ya existente:

```php
$p = new Prueba;

Yii::configure($p, [
    'uno' => 5,
    'dos' => 7,
]);

echo $p->uno; // Muestra "5"
echo $p->dos; // Muestra "7"
```

CREACIÓN DE NUEVAS INSTANCIAS

- Una configuración también se puede usar para crear nuevas instancias y asignarle valores iniciales *en la misma operación* usando el método `Yii::createObject($config)`.
- Para ello es necesario que la configuración indique el nombre de la clase que se desea instanciar mediante un elemento con clave `'class'`.
- Ejemplo:

```php
$p = Yii::createObject([
    'class' => 'Prueba',
    'uno' => 4,
    'dos' => 7,
]);
```

- Se crea en `$p` una nueva instancia de la clase `Prueba` con los valores `$p->uno = 4` y `$p->dos = 7`.

`\yii\base\Component`

- Todos los componentes de Yii 2 heredan, directa o indirectamente, de esta clase.
- Esta clase hereda, a su vez, de `\yii\base\BaseObject`, por lo que incorpora *propiedades* y *configurabilidad*.
- Además, los componentes incorporan otras dos características importantes:

Eventos

Comportamientos

EVENTOS

EVENTOS

- Los eventos permiten inyectar nuestro código dentro de código ya existente en determinados puntos de ejecución.
- Se puede vincular un trozo de código a un evento de forma que, cuando el evento se dispare, se ejecutará el código automáticamente.
- Por ejemplo, un objeto que envíe correo puede disparar el evento `mensajeEnviado` cada vez que envíe un email. Si se desea hacer un seguimiento de los mensajes que se han enviado, se puede vincular el código de seguimiento al evento `mensajeEnviado`.

- Los eventos son un mecanismo que nos permite cambiar el comportamiento del *framework* sin tener que cambiar el código del propio *framework*.
- Esto es así porque el *framework* dispara ciertos eventos en ciertos momentos durante su ejecución, lo que podemos usar para vincular nuestro código y hacer que se ejecute en tales momentos.

Para poder disparar eventos o responder a eventos, la clase en cuestión debe ser subclase (directa o indirecta) de `\yii\base\Component`.

MANEJADORES DE EVENTOS

- Un manejador de eventos es un *callable* de PHP que se ejecutará cuando se dispare el evento al que se haya vinculado.
- El *callable* puede ser:
  - Una función global de PHP especificada en forma de cadena (sin paréntesis): `'trim'`
  - Un método de instancia especificado como un array donde el primer elemento es el objeto y el segundo es el nombre del método como cadena (sin paréntesis): `[$objeto, 'metodo']`
  - Un método estático de clase especificado como un array donde el primer elemento es el nombre de la clase y el segundo es el nombre del método como cadena (sin paréntesis): `['NombreClase', 'metodo']`
  - Una función anónima: `function ($event) { ... }`

La signatura del manejador de eventos es:

```php
function ($event) {
    // $event es un objeto de la clase \yii\base\Event
    // (o una subclase de esta)
}
```

Ejercicio: consultar qué información contiene el objeto `$event`.

VINCULAR MANEJADORES A EVENTOS

Para **vincular un manejador a un evento en un objeto** se usa el método `\yii\base\Component::on()` sobre ese objeto:

```php
$p = new Prueba;

// Este manejador es una función global de PHP:
$p->on(Prueba::EVENTO_HOLA, 'funcion');

// Este manejador es un método de instancia:
$p->on(Prueba::EVENTO_HOLA, [$objeto, 'metodo']);

// Este manejador es un método estático de clase:
$p->on(Prueba::EVENTO_HOLA, ['\app\components\Pepe', 'metodo']);

// Este manejador es una función anónima:
$p->on(Prueba::EVENTO_HOLA, function ($event) {
    // Código que gestiona el evento
});
```

Cuando el objeto dispare el evento, se ejecutará el manejador vinculado en ese evento.

DISPARAR EVENTOS

- Los eventos se disparan llamando al método `\yii\base\Component::trigger()` sobre el objeto que envía (o *dispara*) el evento:

```
$p->trigger(Prueba::EVENTO_HOLA);
```

- El objeto `$p` dispara el evento `Prueba::EVENTO_HOLA`, lo que provocará la ejecución de los manejadores que se hayan vinculado a ese evento en ese objeto (además de los *manejadores de clase*, que veremos luego).
- Si hay varios manejadores para el mismo evento, se ejecutarán en cadena en el orden en el que hayan sido vinculados.
- Un manejador puede hacer `$event->handled = true;` para romper la cadena y evitar que se ejecuten más manejadores de ese evento.

Los manejadores vinculados en otro objeto no se ejecutarán:

```php
$p = new Prueba;
$q = new Prueba;

// $p registra un manejador para el evento Prueba::EVENTO_HOLA:
$p->on(Prueba::EVENTO_HOLA, function ($event) {
    echo "Soy p";
});

// $q registra otro manejador para el mismo evento:
$q->on(Prueba::EVENTO_HOLA, function ($event) {
    echo "Soy q";
});

$p->trigger(Prueba::EVENTO_HOLA); // Muestra "Soy p" pero no "Soy q"
```

Aquí el evento lo dispara `$p` y por tanto no se ejecuta el manejador registrado por `$q` (el objeto `$q` no se entera).

CONSTANTES PARA LOS EVENTOS

- Es recomendable usar constantes de clase para representar los nombres de los eventos.
- En el ejemplo anterior, la constante `Prueba::EVENTO_HOLA` representa el evento `hola`.
- Esto tiene tres ventajas:
  1. Previene equivocaciones al teclear.
  2. Los hace más reconocible por el autocompletado de los editores.
  3. Resulta más fácil saber qué eventos soporta una clase simplemente mirando las constantes que tenga declaradas.

MANEJADORES DE CLASE

- Hasta ahora hemos vinculado manejadores a eventos *a nivel de instancia*, es decir, en objetos concretos.
- A veces, queremos que *todas* las instancias de una clase respondan de la misma forma a un determinado evento.
- En lugar de vincular un manejador de evento en cada instancia, podemos vincular el manejador *a nivel de clase*, es decir, en la propia clase.
- A estos eventos se los denomina **manejadores de clase**, a diferencia de los manejadores a nivel de instancia, que se denominan **manejadores de instancia**.
- Para ello, usamos el método estático `\yii\base\Event::on()` indicando el nombre de la clase, el nombre del evento y el manejador del evento.

Ejemplo:

```php
use yii\base\Event;

Event::on(Prueba::className(), Prueba::EVENTO_HOLA, function ($event) {
    echo "Hola";
});
```

A partir de ahora, todas las instancias (actuales y futuras) de la clase `Prueba` (y sus subclases) responderán al evento `Prueba::EVENTO_HOLA` mostrando un saludo en pantalla:

```php
$a = new Prueba;
$b = new Prueba;
$a->trigger(Prueba::EVENTO_HOLA);  // Saluda
$b->trigger(Prueba::EVENTO_HOLA);  // También saluda
```

TAMBIÉN AFECTA A LAS SUBCLASES

- Los manejadores a nivel de clase se ejecutarán cuando se produzca un evento disparado por cualquier instancia de esa clase **o de cualquier subclase suya**.
- Dicho de otra forma: el evento se propagará hacia arriba en la jerarquía de herencia y provocará la ejecución de todos los manejadores de clase que encuentre vinculados a ese evento.
- Debido a eso, hay que tener cuidado para evitar la propagación de ese evento a más objetos de los necesarios.
- Por ejemplo, si vinculamos un manejador de evento a la clase `\yii\base\BaseObject`, prácticamente *todas* las instancias de Yii 2 podrán responder a ese evento.

Ejemplo:

```php
class Subclase extends Prueba
{
    // ...
}

$s = new Subclase;
$s->trigger(Prueba::EVENTO_HOLA);  // También saluda
```

Una instancia de `Subclase` dispara un evento que tiene vinculado un manejador en la clase `Prueba`, por lo que también se ejecuta dicho manejador.

En los manejadores de clase, se puede acceder al objeto que ha recibido el disparo del evento mediante la expresión `$event->sender`:

```php
use yii\base\Event;

Event::on(Prueba::className(), Prueba::EVENTO_HOLA, function ($event) {
    // $event->sender es el objeto que ha recibido el evento
});
```

ORDEN DE LOS MANEJADORES

- El mismo evento puede tener vinculados manejadores de instancia y manejadores de clase.
- En ese caso, al dispararse el evento se ejecutarán primero los manejadores de instancia y después los manejadores de clase (los manejadores de instancia siempre van primero).

EVENTOS DE CLASE

- Hasta ahora, los eventos han sido disparados por objetos concretos. A estos eventos se los denomina **eventos de instancia**.
- Podemos hacer que una clase también dispare eventos. Los eventos disparados por una clase se denominan **eventos de clase**.
- Los eventos de clase se disparan llamando al método `\yii\base\Event::trigger()`, indicando el nombre de la clase y el nombre del evento.
- Los eventos de clase sólo provocan la ejecución de manejadores de clase, no de manejadores de instancia.

Ejemplo:

```php
use yii\base\Event;

Event::on(Prueba::className(), Prueba::EVENTO_HOLA, function ($event) {
    var_dump($event->sender);  // Muestra "null"
});

// Aquí se dispara el evento de clase.
// Lo dispara directamente la clase Prueba, no una instancia concreta:
Event::trigger(Prueba::className(), Prueba::EVENT_HELLO);
```

Observa que, en este caso, `$event->sender` es `null`, ya que quien dispara el evento no es ninguna instancia concreta, sino una clase.

RESUMIENDO

- Los eventos de instancia provocan la ejecución de manejadores de instancia y manejadores de clase.
- Los eventos de clase sólo provocan la ejecución de manejadores de clase.

COMPORTAMIENTOS

COMPORTAMIENTOS

- Los comportamientos (o *behaviors*) permiten ampliar la funcionalidad de una clase sin afectar a su herencia.
- En otros lenguajes de programación se denominan *mixins*.
- Al *acoplar* un comportamiento a un componente se *inyectan* los métodos y las propiedades del comportamiento dentro del componente.
- El componente podrá usar esos métodos y propiedades como si estuvieran definidos en la clase del componente.
- Además, un comportamiento puede responder a los eventos disparados por el componente, lo que le permite alterar la ejecución normal del código del componente.

DEFINICIÓN DE COMPORTAMIENTOS

Un comportamiento es una subclase (directa o indirecta) de `\yii\base\Behavior`:

```php
class Comportamiento extends \yii\base\Behavior
{
    public $prop1;
    private $_prop2;

    public function pepe()
    {
        // ...
    }

    public function getProp2()
    {
        return $this->_prop2;
    }

    public function setProp2($valor)
    {
        $this->_prop2 = $valor;
    }
}
```

- Este comportamiento define dos propiedades (`prop1` y `prop2`) y un método (`pepe()`).
- Cuando se acople a un componente, éste dispondrá de esas propiedades y ese método.

ACOPLAR COMPORTAMIENTOS A UN COMPONENTE

- Los comportamientos se pueden acoplar de forma *estática* o *dinámica*.
- El acoplamiento estático es el más usado.

ACOPLAMIENTO ESTÁTICO DE COMPONENTES

- Se sobreescribe el método `behaviors()` del componente al que se desea acoplar el comportamiento.
- El método `behaviors()` debe devolver un array de configuraciones de comportamientos.
- Cada configuración puede ser:
  - El nombre de una clase comportamiento, o
  - Un array de configuración.

Ejemplo de acoplamiento estático:

```php
class Usuario extends \yii\db\ActiveRecord
{
    public function behaviors()
    {
        return [
            // anónimo, sólo el nombre de la clase
            Comportamiento::className(),

            // con nombre, sólo el nombre de la case
            'comp2' => Comportamiento::className(),

            // anónimo, array de configuración
            [
                'class' => Comportamiento::className(),
                'prop1' => 'valor1',
                'prop2' => 'valor2',
            ],

            // con nombre, array de configuración
            'comp4' => [
                'class' => Comportamiento::className(),
                'prop1' => 'valor1',
                'prop2' => 'valor2',
            ],
        ];
    }
}
```

- Se le puede asociar un nombre a un comportamiento especificándolo en el array en la clave correspondiente a la configuración de ese comportamiento.
- En tal caso, al comportamiento se le denomina **comportamiento con nombre**.

- En el ejemplo anterior, hay dos comportamientos con nombre: `comp2` y `comp4`.
- Si un comportamiento no lleva asociado ningún nombre, se le denomina **comportamiento anónimo**.

ACOPLAMIENTO DINÁMICO DE COMPORTAMIENTOS

Para acoplar un comportamiento dinámicamente, se llama al método `\yii\base\Component::attachBehavior()` method del componente al que se le va a acoplar el comportamiento:

```php
// acopla un objeto comportamiento
$componente->attachBehavior('comp1', new Comportamiento);

// acopla una clase comportamiento
$componente->attachBehavior('comp2', Comportamiento::className());

// acopla un array de configuración
$componente->attachBehavior('comp3', [
    'class' => Comportamiento::className(),
    'prop1' => 'valor1',
    'prop2' => 'valor2',
]);
```

USO DE COMPORTAMIENTOS

- Para usar un comportamiento, primero hay que acoplarlo a un componente usando uno de los métodos que se han visto antes.
- A partir de ese momento, el componente podrá:
  - Acceder a cualquier variable de instancia pública o propiedad pública definida en el comportamiento como si estuvieran realmente definidas en el componente.
  - Invocar a cualquier método público definido en el comportamiento como si estuviera definido en el componente.

Ejemplo:

```php
// "prop1" es una propiedad definida en la clase Comportamiento
echo $componente->prop1;
$componente->prop1 = $valor;

// pepe() es un método definido en la clase Comportamiento
$componente->pepe();
```

Como se ve, aunque `$componente` no tiene definda la propiedad `prop1` ni el método `pepe()`, puede usarlos como si fueran parte de la definición del componente gracias a que tiene acoplado el comportamiento `Comportamiento`.