CONCEPTOS FUNDAMENTALES DE YUI 2

Ricardo Pérez López

IES Doñana, curso 2017-18

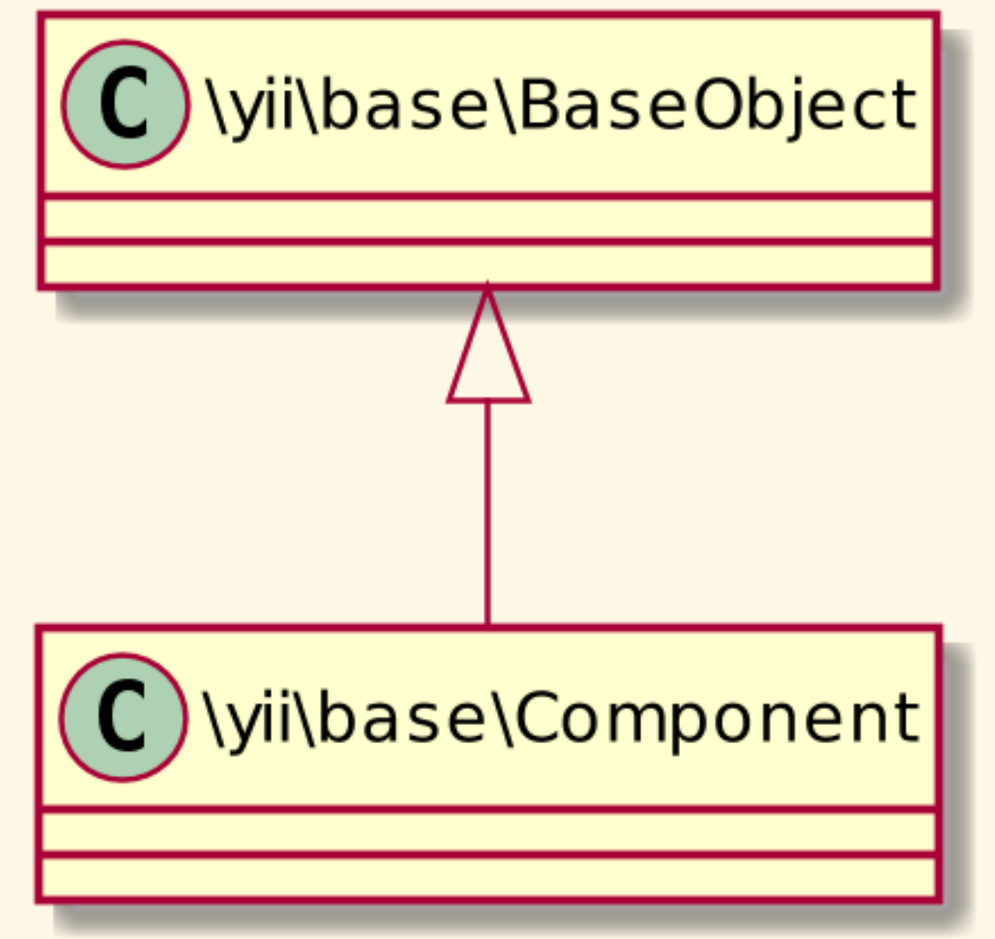
COMPONENTES

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

- 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

- 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

- Los eventos permiten inyectar código propio 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.

Si una clase necesita disparar eventos o responder a eventos, necesita 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 vaya asociado.
- 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 se usa el método `\yii\base\Component::on()`:

```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
});
```

DISPARAR EVENTOS

- Los eventos se disparan llamando al método `\yii\base\Component::trigger()`:

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

- El evento le llega al objeto sobre el que se ejecuta el método `trigger()`, y responderá ejecutando los manejadores que el objeto tenga vinculados al evento.
- Si hay varios manejadores para el mismo evento, se ejecutarán en el orden en el que hayan sido vinculados.

- 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, a instancias concretas.
- 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 a cada instancia, podemos vincular el manejador *a nivel de clase*, es decir, a la propia clase.
- Para ello, usamos el método estático `\yii\base\Event::on()`.

Ejemplo:

```php
use \yii\base\Event;

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