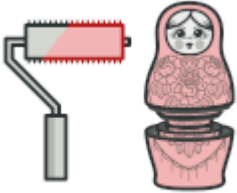




[🏠](#) / [Patrones de diseño](#) / [Decorator](#) / [PHP](#)



# Decorator en PHP

**Decorator** es un patrón de diseño estructural que permite añadir dinámicamente nuevos comportamientos a objetos colocándolos dentro de objetos especiales que los envuelven (*\_wrappers\_*).

Utilizando decoradores puedes envolver objetos innumerables veces, ya que los objetos objetivo y los decoradores siguen la misma interfaz. El objeto resultante obtendrá un comportamiento de apilado de todos los *wrappers*.

 Aprende más sobre el patrón Decorator →

## Navegación

 [Intro](#)

 [Ejemplo conceptual](#)

 [index](#)

 [Output](#)

 [Ejemplo del mundo real](#)

 [index](#)

 [Output](#)

Complejidad: ★★☆☆

Popularidad: ★★☆☆

**Ejemplos de uso:** El patrón Decorator es bastante común en el código PHP, especialmente en el código relacionado con los flujos (streams).



## Ejemplo conceptual

Este ejemplo ilustra la estructura del patrón de diseño **Decorator** y se centra en las siguientes preguntas:

- ¿De qué clases se compone?
- ¿Qué papeles juegan esas clases?
- ¿De qué forma se relacionan los elementos del patrón?

Después de conocer la estructura del patrón, será más fácil comprender el siguiente ejemplo basado en un caso de uso real de PHP.

### index.php: Ejemplo conceptual

```
<?php

namespace RefactoringGuru\Decorator\Conceptual;

/**
 * The base Component interface defines operations that can be altered by
 * decorators.
 */
interface Component
{
    public function operation(): string;
}

/**
 * Concrete Components provide default implementations of the operations. There
 * might be several variations of these classes.
 */
class ConcreteComponent implements Component
{
    public function operation(): string
    {
        return "ConcreteComponent";
    }
}

/**
 * The base Decorator class follows the same interface as the other components.
```



```
* include a field for storing a wrapped component and the means to initialize
* it.
*/
class Decorator implements Component
{
    /**
     * @var Component
     */
    protected $component;

    public function __construct(Component $component)
    {
        $this->component = $component;
    }

    /**
     * The Decorator delegates all work to the wrapped component.
     */
    public function operation(): string
    {
        return $this->component->operation();
    }
}

/**
 * Concrete Decorators call the wrapped object and alter its result in some way.
 */
class ConcreteDecoratorA extends Decorator
{
    /**
     * Decorators may call parent implementation of the operation, instead of
     * calling the wrapped object directly. This approach simplifies extension
     * of decorator classes.
     */
    public function operation(): string
    {
        return "ConcreteDecoratorA(" . parent::operation() . ")";
    }
}

/**
 * Decorators can execute their behavior either before or after the call to a
 * wrapped object.
 */
class ConcreteDecoratorB extends Decorator
{
    public function operation(): string
    {
        return "ConcreteDecoratorB(" . parent::operation() . ")";
    }
}
```



```
/**
 * The client code works with all objects using the Component interface. This
 * way it can stay independent of the concrete classes of components it works
 * with.
 */
function clientCode(Component $component)
{
    // ...

    echo "RESULT: " . $component->operation();

    // ...
}

/**
 * This way the client code can support both simple components...
 */
$simple = new ConcreteComponent();
echo "Client: I've got a simple component:\n";
clientCode($simple);
echo "\n\n";

/**
 * ...as well as decorated ones.
 *
 * Note how decorators can wrap not only simple components but the other
 * decorators as well.
 */
$decorator1 = new ConcreteDecoratorA($simple);
$decorator2 = new ConcreteDecoratorB($decorator1);
echo "Client: Now I've got a decorated component:\n";
clientCode($decorator2);
```

## Output.txt: Resultado de la ejecución

```
Client: I've got a simple component:
RESULT: ConcreteComponent
```

```
Client: Now I've got a decorated component:
RESULT: ConcreteDecoratorB(ConcreteDecoratorA(ConcreteComponent))
```

## Ejemplo del mundo real



para limpiar el contenido antes de representarlo en una página web. Los distintos tipos de contenido, como comentarios, publicaciones en foros, o mensajes privados, requieren distintos grupos de filtros.

Por ejemplo, aunque quieras eliminar todo el HTML de los comentarios, quizá desees mantener algunas etiquetas HTML básicas en publicaciones en el foro. Además, puede que quieras permitir publicar en formato Markdown, que debe procesarse antes de que se realice el filtrado HTML. Todas estas reglas de filtrado pueden representarse como clases decoradoras separadas, que se pueden apilar de otra forma, dependiendo de la naturaleza del contenido que tengas.

## index.php: Ejemplo del mundo real

```
<?php

namespace RefactoringGuru\Decorator\RealWorld;

/**
 * The Component interface declares a filtering method that must be implemented
 * by all concrete components and decorators.
 */
interface InputFormat
{
    public function formatText(string $text): string;
}

/**
 * The Concrete Component is a core element of decoration. It contains the
 * original text, as is, without any filtering or formatting.
 */
class TextInput implements InputFormat
{
    public function formatText(string $text): string
    {
        return $text;
    }
}

/**
 * The base Decorator class doesn't contain any real filtering or formatting
 * logic. Its main purpose is to implement the basic decoration infrastructure:
 * a field for storing a wrapped component or another decorator and the basic
 * formatting method that delegates the work to the wrapped object. The real
 * formatting job is done by subclasses.
 */
class TextFormat implements InputFormat
{

```



```
*/
protected $inputFormat;

public function __construct(InputFormat $inputFormat)
{
    $this->inputFormat = $inputFormat;
}

/**
 * Decorator delegates all work to a wrapped component.
 */
public function formatText(string $text): string
{
    return $this->inputFormat->formatText($text);
}
}

/**
 * This Concrete Decorator strips out all HTML tags from the given text.
 */
class PlainTextFilter extends TextFormat
{
    public function formatText(string $text): string
    {
        $text = parent::formatText($text);
        return strip_tags($text);
    }
}

/**
 * This Concrete Decorator strips only dangerous HTML tags and attributes that
 * may lead to an XSS vulnerability.
 */
class DangerousHTMLTagsFilter extends TextFormat
{
    private $dangerousTagPatterns = [
        "|<script.*?>([\s\S]*)?</script>|i", // ...
    ];

    private $dangerousAttributes = [
        "onclick", "onkeypress", // ...
    ];

    public function formatText(string $text): string
    {
        $text = parent::formatText($text);

        foreach ($this->dangerousTagPatterns as $pattern) {
            $text = preg_replace($pattern, '', $text);
        }
    }
}
```



# REBAJA DE VERANO



```
foreach ($this->dangerousAttributes as $attribute) {
    $text = preg_replace_callback('<|<(.*?)>|', function ($matches) use ($attribut
        $result = preg_replace("|$attribute=|i", '', $matches[1]);
        return "<" . $result . ">";
    }, $text);
}

return $text;
}
}

/**
 * This Concrete Decorator provides a rudimentary Markdown → HTML conversion.
 */
class MarkdownFormat extends TextFormat
{
    public function formatText(string $text): string
    {
        $text = parent::formatText($text);

        // Format block elements.
        $chunks = preg_split('|\\n\\n|', $text);
        foreach ($chunks as &$chunk) {
            // Format headers.
            if (preg_match('|^#+|', $chunk)) {
                $chunk = preg_replace_callback('|^(#+)(.*?)$|', function ($matches) {
                    $h = strlen($matches[1]);
                    return "<h$h>" . trim($matches[2]) . "</h$h>";
                }, $chunk);
            } // Format paragraphs.
            else {
                $chunk = "<p>$chunk</p>";
            }
        }
        $text = implode("\\n\\n", $chunks);

        // Format inline elements.
        $text = preg_replace("|_|(.*?)_|", '<strong>$1</strong>', $text);
        $text = preg_replace("|\\*\\*(.*?)\\*\\*|", '<strong>$1</strong>', $text);
        $text = preg_replace("|_|(.*?)_|", '<em>$1</em>', $text);
        $text = preg_replace("|\\*(.*?)\\*|", '<em>$1</em>', $text);

        return $text;
    }
}

/**
 * The client code might be a part of a real website, which renders user-
 * generated content. Since it works with formatters through the Component
```



```
*/  
function displayCommentAsAWebsite(InputFormat $format, string $text)  
{  
    // ..  
  
    echo $format->formatText($text);  
  
    // ..  
}  
  
/**  
 * Input formatters are very handy when dealing with user-generated content.  
 * Displaying such content "as is" could be very dangerous, especially when  
 * anonymous users can generate it (e.g. comments). Your website is not only  
 * risking getting tons of spammy links but may also be exposed to XSS attacks.  
 */  
$dangerousComment = <<<HERE  
Hello! Nice blog post!  
Please visit my <a href='http://www.iwillhackyou.com'>homepage</a>.  
<script src="http://www.iwillhackyou.com/script.js">  
    performXSSAttack();  
</script>  
HERE;  
  
/**  
 * Naive comment rendering (unsafe).  
 */  
$naiveInput = new TextInput();  
echo "Website renders comments without filtering (unsafe):\n";  
displayCommentAsAWebsite($naiveInput, $dangerousComment);  
echo "\n\n\n";  
  
/**  
 * Filtered comment rendering (safe).  
 */  
$filteredInput = new PlainTextFilter($naiveInput);  
echo "Website renders comments after stripping all tags (safe):\n";  
displayCommentAsAWebsite($filteredInput, $dangerousComment);  
echo "\n\n\n";  
  
/**  
 * Decorator allows stacking multiple input formats to get fine-grained control  
 * over the rendered content.  
 */  
$dangerousForumPost = <<<HERE  
# Welcome  
  
This is my first post on this **gorgeous** forum.
```





```
</script>
HERE;

/**
 * Naive post rendering (unsafe, no formatting).
 */
$naiveInput = new TextInput();
echo "Website renders a forum post without filtering and formatting (unsafe, ugly):\n";
displayCommentAsAWebsite($naiveInput, $dangerousForumPost);
echo "\n\n\n";

/**
 * Markdown formatter + filtering dangerous tags (safe, pretty).
 */
$text = new TextInput();
$markdown = new MarkdownFormat($text);
$filteredInput = new DangerousHTMLTagsFilter($markdown);
echo "Website renders a forum post after translating markdown markup" .
    " and filtering some dangerous HTML tags and attributes (safe, pretty):\n";
displayCommentAsAWebsite($filteredInput, $dangerousForumPost);
echo "\n\n\n";
```

## Output.txt: Resultado de la ejecución

```
Website renders comments without filtering (unsafe):
Hello! Nice blog post!
Please visit my <a href='http://www.iwillhackyou.com'>homepage</a>.
<script src="http://www.iwillhackyou.com/script.js">
    performXSSAttack();
</script>
```

```
Website renders comments after stripping all tags (safe):
Hello! Nice blog post!
Please visit my homepage.
```

```
    performXSSAttack();
```

```
Website renders a forum post without filtering and formatting (unsafe, ugly):
# Welcome
```

```
This is my first post on this **gorgeous** forum.
```

```
<script src="http://www.iwillhackyou.com/script.js">
```



# REBAJA DE VERANO



Website renders a forum post after translating markdown markup and filtering some dangerous HTML tags.

```
<h1>Welcome</h1>
```

```
<p>This is my first post on this <strong>gorgeous</strong> forum.</p>
```

[Inicio](#)



[Refactorización](#)



[Patrones de diseño](#)



[Contenido Premium](#)

[Foro](#)


[Contáctanos](#)

© 2014-2023 Refactoring.Guru. Todos los derechos reservados

 Ilustraciones por Dmitry Zhart

#### Ukrainian office:

 FOP Olga Skobeleva


 Abolmasova 7

Kyiv, Ukraine, 02002

 Email: [support@refactoring.guru](mailto:support@refactoring.guru)

#### Spanish office:

 Oleksandr Shvets

 Avda Pamplona 63, 4b

Pamplona, Spain, 31010

 Email: [spain@refactoring.guru](mailto:spain@refactoring.guru)

[Términos y condiciones](#)

[Política de privacidad](#)

[Política de uso de contenido](#)

[About us](#)

