

## PATRÓN DECORATOR



Laboratorio de Ingeniería de Software II

Presentado por:

Ledy Astudillo

Santiago Nieto Guaca

Profesor:

Ricardo Zambrano

**Universidad del Cauca**

Facultad de Ingeniería Electrónica y telecomunicaciones

Ingeniería de sistemas

Popayán, octubre 2023

1. Desarrolle la siguiente plantilla para el patrón Decorador:

Patrón creacional: <b>Decorador</b>	
<b>Intención</b>	El patrón de diseño Decorador tiene la intención de agregar dinámicamente responsabilidades adicionales a un objeto. Proporciona una alternativa flexible a la herencia para extender la funcionalidad.
<b>Problema que soluciona</b>	Cuando se necesita agregar responsabilidades a objetos individuales, no a una clase entera, y cuando las extensiones de responsabilidad deben ser aplicables en tiempo de ejecución.
<b>Solución propuesta</b>	Implica un conjunto de clases decoradoras que se utilizan para envolver objetos concretos. Las clases decoradoras reflejan el tipo de objetos concretos que decoran. Heredan del objeto que decoran.
<b>Diagrama de clases</b>	<pre>classDiagram     class IComponent {         &lt;&lt;Interface&gt;&gt;         +decorate()     }     class ConcreteComponent {         +decorate()     }     class ComponentDecorator {         &lt;&lt;Abstract&gt;&gt;         +decorate()     }     class ComponentDecoratorImplA {         +decorate()         +customDecorateA()     }     class ComponentDecoratorB {         +decorate()         +customDecorateB()     }     IComponent &lt; -- ConcreteComponent     IComponent &lt; -- ComponentDecorator     ComponentDecorator &lt; -- ComponentDecoratorImplA     ComponentDecorator &lt; -- ComponentDecoratorB     ComponentDecorator "1" *-- IComponent</pre> <p>The diagram illustrates the Decorator Pattern structure. At the top is the <b>IComponent</b> interface, which defines the <code>+ decorate()</code> method. Below it, <b>ConcreteComponent</b> and <b>ComponentDecorator</b> (an abstract class) both extend <b>IComponent</b>, as indicated by the 'Extends' labels and hollow triangle arrows. <b>ConcreteComponent</b> also implements the <code>+ decorate()</code> method. <b>ComponentDecorator</b> is an abstract class that also implements <code>+ decorate()</code> and holds a reference to an <b>IComponent</b> object (indicated by a solid line with an open diamond and a '1' multiplicity). Below <b>ComponentDecorator</b>, <b>ComponentDecoratorImplA</b> and <b>ComponentDecoratorB</b> both extend <b>ComponentDecorator</b>. <b>ComponentDecoratorImplA</b> implements <code>+ decorate()</code> and <code>+ customDecorateA()</code>, while <b>ComponentDecoratorB</b> implements <code>+ decorate()</code> and <code>+ customDecorateB()</code>.</p>

<p><b>Diagrama de secuencia</b></p>	<pre> sequenceDiagram     participant Client     participant ComponentDecoratorA     participant ComponentDecoratorB     participant ConcreteComponent      Client-&gt;&gt;ComponentDecoratorA: decorate()     activate ComponentDecoratorA     ComponentDecoratorA-&gt;&gt;ComponentDecoratorB: decorate()     activate ComponentDecoratorB     ComponentDecoratorB-&gt;&gt;ConcreteComponent: decorate()     activate ConcreteComponent     ConcreteComponent--&gt;&gt;ComponentDecoratorB: return     deactivate ConcreteComponent     ComponentDecoratorB--&gt;&gt;ComponentDecoratorA: return     deactivate ComponentDecoratorB     ComponentDecoratorA--&gt;&gt;Client: return     deactivate ComponentDecoratorA   </pre>
<p><b>Participantes</b></p>	<ul style="list-style-type: none"> <li>● <b>Componente:</b> Define la interfaz para los objetos que pueden tener responsabilidades añadidas dinámicamente.</li> <li>● <b>Concreto Componente:</b> Define un objeto al que se pueden añadir responsabilidades adicionales.</li> <li>● <b>Decorador:</b> Mantiene una referencia a un objeto Componente y define una interfaz que cumple con la interfaz del Componente.</li> <li>● <b>Concreto Decorador:</b> Añade responsabilidades al Componente.</li> </ul>
<p><b>Aplicabilidad</b></p>	<ol style="list-style-type: none"> <li>1. Se necesita añadir responsabilidades a objetos individuales de manera dinámica y transparente, es decir, sin afectar a otros objetos.</li> <li>2. Se necesita retirar las responsabilidades de un objeto.</li> <li>3. La extensión mediante la herencia no es viable. En muchos casos, una cantidad excesiva de opciones independientes entre sí conduciría a una explosión de subclases para cubrir todas las combinaciones posibles.</li> </ol>
<p><b>Consecuencias</b></p>	<ol style="list-style-type: none"> <li>1. Más flexibilidad que la herencia estática.</li> <li>2. Evita la carga de “clases con características”, es decir, clases altamente personalizadas al final del árbol de herencia.</li> <li>3. Un decorador y su componente no son idénticos.</li> <li>4. Muchos pequeños objetos pueden resultar difíciles de aprender y depurar.</li> </ol>

## REFERENCIAS

- [1] “Patrones de Diseño (X): Patrones Estructurales - Decorator”. Programación en Castellano. Accedido el 16 de octubre de 2023. [En línea]. Disponible: <https://tinyurl.com/67k7a5xh>
- [2] Reactive Programming - arquitectura y desarrollo de software. Accedido el 16 de octubre de 2023. [En línea]. Disponible: <https://tinyurl.com/54ka9shn>