

## PATRON COMPOSITE



Laboratorio de Ingeniería de Software II

Presentado por:

Ledy Astudillo

Santiago Nieto Guaca

Docente:

Ricardo Zambrano Segura

***Universidad del Cauca***

Facultad de Ingeniería Electrónica y telecomunicaciones

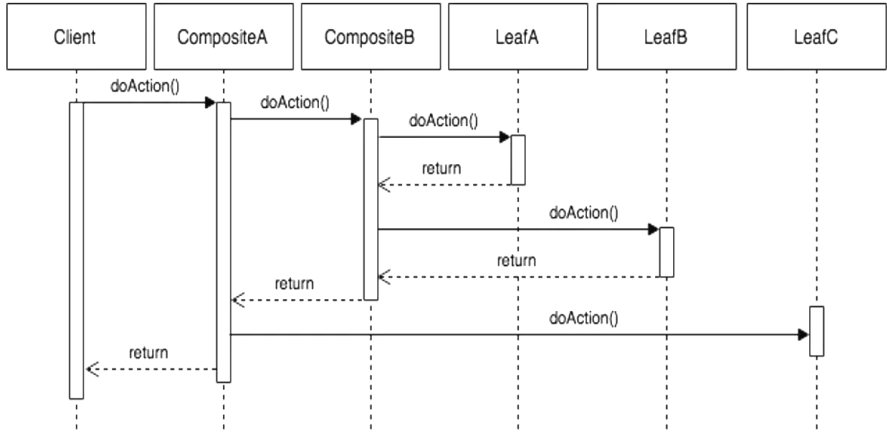
Ingeniería de Sistemas

Popayán

2023

1. Desarrolle la siguiente plantilla para el patrón Composite:

Patrón estructural: Composite	
Intención	Organizar objetos en una estructura de árbol para representar relaciones de jerarquía de parte y todo. Además proporciona la capacidad de tratar tanto a los objetos individuales como a las agrupaciones de objetos de la misma forma.
Problema que soluciona	Cuando los programadores trabajan con datos que están organizados en una estructura de árbol, a menudo se encuentran en la necesidad de diferenciar entre un nodo hoja y un nodo que es una rama. Esta necesidad puede complicar el código que se escribe para el cliente.
Solución propuesta	Define una interfaz unificada (Componente) para los objetos parte (Hoja) y los objetos todo (Composite). Los objetos Hoja implementan la interfaz Componente directamente, y los objetos Composite reenvían las solicitudes a sus componentes hijos.
Diagrama de clases	<pre>classDiagram     class Client     class Component {         &lt;&lt;interface&gt;&gt;         +execute()     }     class Leaf {         ...         +execute()     }     class Composite {         -children: Component[]         +add(c: Component)         +remove(c: Component)         +getChildren(): Component[]         +execute()     }     Client --&gt; Component     Component &lt; .. Leaf     Component o-- Composite</pre> <p>The diagram illustrates the Composite Pattern structure. It features a <b>Client</b> class that interacts with a <b>Component</b> interface. The <b>Component</b> interface defines an <code>+ execute()</code> method. Two classes, <b>Leaf</b> and <b>Composite</b>, implement this interface. <b>Leaf</b> is a simple class with an <code>+ execute()</code> method, annotated with the note "Hacer algo de trabajo." (Do some work). <b>Composite</b> is a more complex class that maintains an array of <b>Component</b> objects (labeled <code>- children: Component[]</code>). It implements <code>+ execute()</code> by delegating the request to its children, as noted: "Delegar todo el trabajo a componentes hijos." (Delegate all work to child components). The <b>Composite</b> class also provides methods for managing its children: <code>+ add(c: Component)</code>, <code>+ remove(c: Component)</code>, and <code>+ getChildren(): Component[]</code>. A solid arrow points from <b>Client</b> to <b>Component</b>. A dashed arrow with a hollow triangle head points from <b>Leaf</b> to <b>Component</b>, indicating inheritance. A solid line with a hollow diamond head points from <b>Composite</b> to <b>Component</b>, indicating aggregation.</p>

<p>Diagrama de secuencia</p>	 <pre> sequenceDiagram     participant Client     participant CompositeA     participant CompositeB     participant LeafA     participant LeafB     participant LeafC      Client-&gt;&gt;CompositeA: doAction()     activate CompositeA     CompositeA-&gt;&gt;CompositeB: doAction()     activate CompositeB     CompositeB-&gt;&gt;LeafA: doAction()     activate LeafA     LeafA--&gt;&gt;CompositeB: return     deactivate LeafA     CompositeB-&gt;&gt;LeafB: doAction()     activate LeafB     LeafB--&gt;&gt;CompositeB: return     deactivate LeafB     CompositeB--&gt;&gt;CompositeA: return     deactivate CompositeB     CompositeA-&gt;&gt;LeafC: doAction()     activate LeafC     LeafC--&gt;&gt;CompositeA: return     deactivate LeafC     CompositeA--&gt;&gt;Client: return     deactivate CompositeA </pre>
<p>Participantes</p>	<p>Los participantes en el patrón Composite incluyen el Componente, la Hoja, el Composite y el Cliente.</p>
<p>Aplicabilidad</p>	<p>Es bastante útil cuando se quiere manejar de manera consistente tanto a los objetos individuales como a los grupos de objetos, y cuando se busca representar relaciones de jerarquía entre un todo y sus partes.</p>
<p>Consecuencias</p>	<p>Hace que el código del cliente sea más sencillo, ya que permite manejar de la misma forma tanto a los objetos individuales como a los grupos de objetos. No obstante, puede resultar en un diseño que es demasiado amplio o general.</p>

## REFERENCIAS BIBLIOGRÁFICAS

[1] “Composite”. Reactive Programming - arquitectura y desarrollo de software. Accedido el 28 de octubre de 2023. [En línea]. Disponible: <https://reactiveprogramming.io/blog/en/design-patterns/composite>

[2] Contributors to Wikimedia projects. “Composite pattern - Wikipedia”. Wikipedia, the free encyclopedia. Accedido el 28 de octubre de 2023. [En línea]. Disponible: [https://en.wikipedia.org/wiki/Composite\\_pattern](https://en.wikipedia.org/wiki/Composite_pattern)

[3] “Composite Design Pattern - GeeksforGeeks”. GeeksforGeeks. Accedido el 28 de octubre de 2023. [En línea]. Disponible: <https://www.geeksforgeeks.org/composite-design-pattern/>