

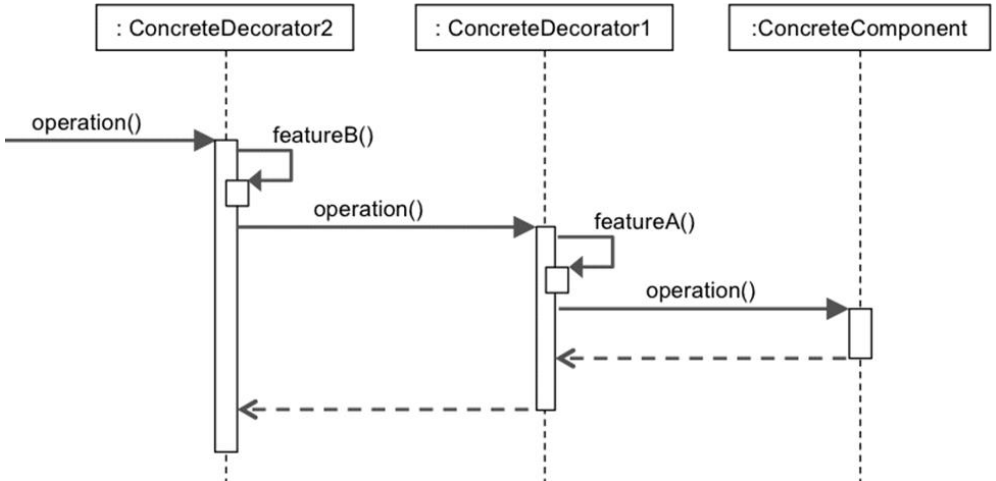
Practica de laboratorio No. 6

Integrantes:

- Daniel Fernando Solarte Ortega
- Cristian David Quinayas Rivera

1.

Patrón creacional: Decorador	
Intención	Agregar funcionalidades a una subclase sin la necesidad de añadir más herencias.
Problema que soluciona	Aplicar muchas herencias para resolver un problema no es siempre la solución correcta ya que esto causa que el diseño del sistema se vuelva más inflexible, o estático, además de agrandar el código en gran medida cuando se quieren combinar funcionalidades.
Solución propuesta	En vez de añadir más herencias por cada funcionalidad que se quiera agregar lo que se hace es que tanto la clase principal como la clase de sus “decoraciones” hereden de una interface, dicha interface utilizara una composición recursiva que permitirá añadir un número ilimitado de funcionalidades a la clase principal permitiendo hacer combinaciones sin la necesidad de crear más subclases por cada combinación de funcionalidades que quiera hacer.
Diagrama de clases	<pre> classDiagram class Client class Component { <<interface>> +execute() } class ConcreteComponent { ... +execute() } class BaseDecorator { -wrappee: Component +BaseDecorator(c: Component) +execute() } class ConcreteDecorators { ... +execute() +extra() } Client --> Component Component < -- ConcreteComponent Component < -- BaseDecorator BaseDecorator o--> Component : wrappee BaseDecorator < -- ConcreteDecorators BaseDecorator ..> BaseDecorator : wrappee.execute() ConcreteDecorators ..> BaseDecorator : super::execute() ConcreteDecorators ..> ConcreteDecorators : extra() </pre> <p>Code snippets from the diagram:</p> <pre> // Client a = new ConcComponent() b = new ConcDecorator1(a) c = new ConcDecorator2(b) c.execute() // Decorator -> Decorator -> Component // BaseDecorator - wrappee: Component + BaseDecorator(c: Component) + execute() wrappee = c // ConcreteDecorators + execute() + extra() super::execute() extra() </pre>

<p>Diagrama de secuencia</p>	 <pre> sequenceDiagram participant Client participant CD2 as : ConcreteDecorator2 participant CD1 as : ConcreteDecorator1 participant CC as : ConcreteComponent Client->>CD2: operation() activate CD2 CD2->>CD2: featureB() deactivate CD2 CD2->>CD1: operation() activate CD1 CD1->>CD1: featureA() deactivate CD1 CD1->>CC: operation() activate CC CC-->>CD1: deactivate CC CD1-->>CD2: deactivate CD1 CD2-->>Client: deactivate CD2 </pre> <p>The diagram illustrates the execution flow of the Decorator Pattern. It involves three lifelines: <code>: ConcreteDecorator2</code>, <code>: ConcreteDecorator1</code>, and <code>: ConcreteComponent</code>. The process begins with an external call to <code>operation()</code> on <code>: ConcreteDecorator2</code>. This decorator then performs <code>featureB()</code> and delegates the <code>operation()</code> call to <code>: ConcreteDecorator1</code>. Similarly, <code>: ConcreteDecorator1</code> performs <code>featureA()</code> and delegates the <code>operation()</code> call to <code>: ConcreteComponent</code>. After the component completes its operation, control returns up the chain of decorators until it reaches the initial caller.</p>
<p>Participantes</p>	<ul style="list-style-type: none"> • Interface encapsuladora. • Clase concreta (implementa la interfaz). • Clase de decoraciones (implementa la interfaz). • Subclases de decoraciones concretas (Hereda de la clase de decoraciones).
<p>Aplicabilidad</p>	<ul style="list-style-type: none"> • Se utiliza cuando se necesita asignar o eliminar funcionalidades a un objeto durante la ejecución sin descomponer el código que utiliza dicho objeto. • Se utiliza cuando aplicar herencia resulta extraño o no sea posible extenderlo con esta técnica.
<p>Consecuencias</p>	<ul style="list-style-type: none"> • Tenga en cuenta que este patrón permite agregar responsabilidades a un objeto, no métodos a la interfaz de un objeto. La interfaz presentada al cliente debe permanecer constante a medida que se especifican capas sucesivas. • La identidad del objeto principal se oculta dentro del objeto decorador, de forma que acceder directamente al objeto principal ahora es algo más complicado. • El orden en que se agregar capas o decoradores concretos tiene relevancia en el resultado final del componente.