PATRÓN VISITOR

Diseño de Sistemas

Manuel Luques
UTN Facultad Regional San Francisco

INFORMACIÓN

En el ámbito de la programación orientada a objetos existen numerosos patrones de diseño, los cuales tienen como fin último proveer soluciones generales a problemas recurrentes en el diseño de software. El patrón que hoy nos compete, es el patrón VISITOR (Visitante).

Propósito

En un contexto de diseño de software, es común el querer añadir funcionalidad a una jerarquía de clases mediante la definición de una operación f(), pero resulta inconveniente tener que modificar cada una de las clases para llevar a cabo su implementación.

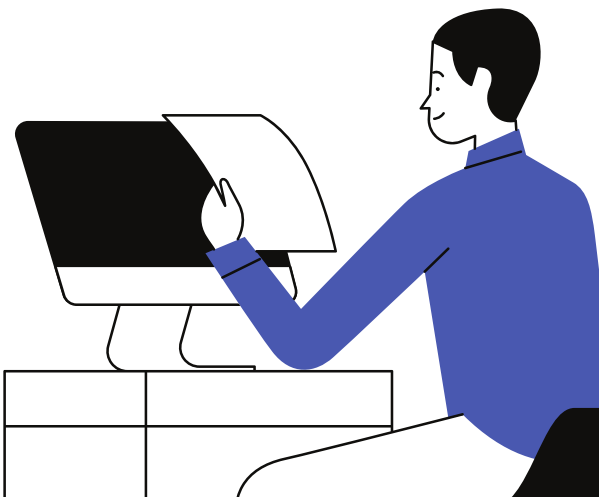
De acuerdo con los principios de diseño SOLID, más específicamente con el principio Abierto/Cerrado, «una entidad de software (clase, módulo, función, etc.) debe quedarse abierta para su extensión, pero cerrada para su modificación». Es decir, se debe poder extender el comportamiento de tal entidad pero sin modificar su código fuente. Aquí es dónde radica la importancia del patrón Visitor.

La finalidad básica del susodicho, es la de separar el algoritmo de la estructura del objeto. Por lo general, se utiliza para definir una operación sobre objetos de una jerarquía de clases sin la necesidad de modificar las mismas.

Aplicabilidad

El mismo es aplicable cuando se desean definir operaciones sobre varias clases de objetos de una estructura jerárquica con interfaces diferentes y la lógica detrás de las mismas depende de la clase concreta con la que se está tratando.

En caso de que la jerarquía sea dinámica, es decir, que cambia; habrá que modificar la estructura de clases para que se adapte a la nueva forma de la estructura jerárquica.

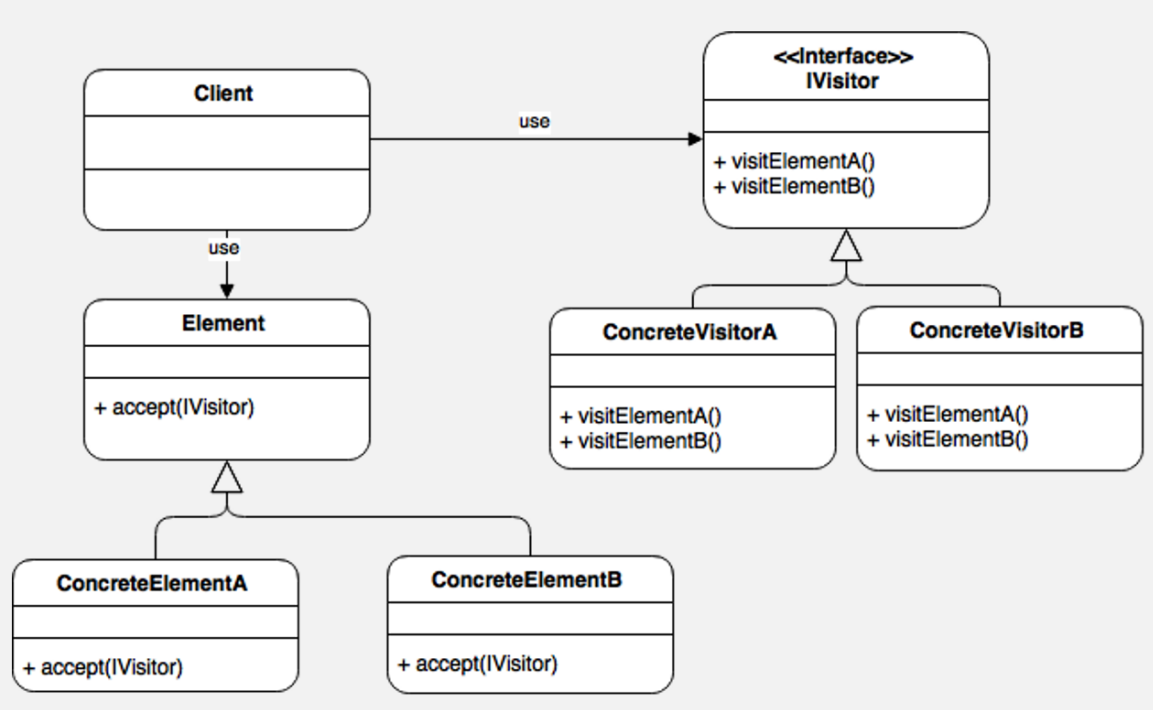


IMPLEMENTACIÓN

En ocasiones nos encontraremos con estructuras de datos sobre las cuales habrá definido un conjunto operaciones variadas el cual crecerá a medida que surjan nuevos requerimientos. Este incremento en el número de operaciones pueden terminar por complejizar de sobremanera dicha estructura.

Por esta razón el patrón de diseño Visitor propone la separación de estas operaciones en clases independientes llamadas Visitantes, las cuales son creadas implementando una interface común y no requiere modificar la estructura inicial para agregar la operación.

Estructura



Clases

Cliente: Así se lo llama al componente que interactúa con la estructura (element) y con el Visitor. Es responsable de crear los visitantes y enviarlos al elemento para su procesamiento.

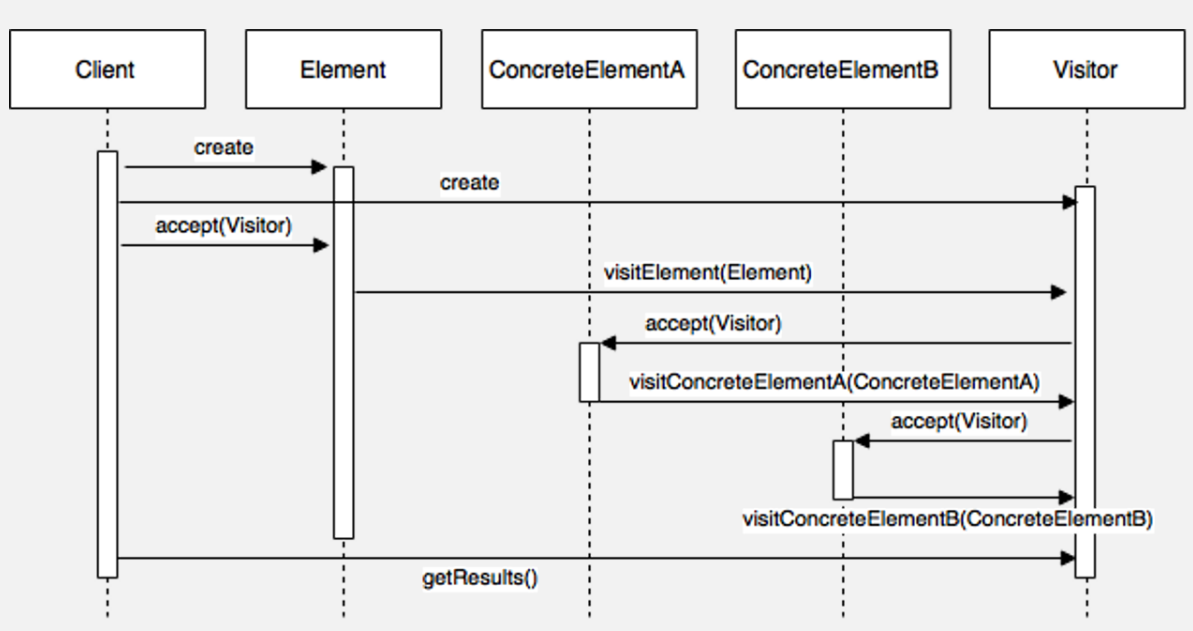
Element: Es la raíz de la estructura sobre la que utilizaremos el Visitor. Por lo general es una interface que define el método accept, el cual deberán implementar todos los objetos de la estructura.

ConcreteElement: Representa un hijo de la estructura compuesta y la misma puede contar un gran número de estos. Todos y cada uno deberá implementar el método accept.

IVisitor: Es la interface donde define la estructura del Visitor. Deberá tener un método *visit* por cada objeto que se requiera analizar de la estructura (element).

ConcreteVisitor: Representa una implementación del Visitor, la cual puede realizar una operación sobre el element. Por cada operación que se quiera añadir, será necesario contar con un ConcreteVisitor donde se defina e implemente la lógica de la operación.

Secuencia



- El cliente crea la estructura (Element).
- El cliente crea la instancia del Visitor a utilizar sobre la estructura.
- El cliente ejecuta el método accept de la estructura y la envía al Visitor.
- El Element le dice al Visitor con que método lo debe procesar. El Visitor deberá tener un método para cada tipo de clase de la estructura.
- El Visitor analiza al Element mediante su método visitElement y repite el proceso de ejecutar el método accept sobre los hijos del Element. Nuevamente el Visitor deberá tener un método para procesar cada clase hija de la estructura.
- El ConcreteElementA le indica al Visitor con qué método debe procesarlo, el cual es visitElementA.
- La Visitor continúa con los demás hijos de Element y esta vez ejecuta el método accept sobre el ConcreteElementB.
- El ConcrteElementB le indica al Visitor con qué método debe procesarlo, el cual es visitElementB.
- Finalmente el Visitor termina la operación sobre la estructura cuando ha recorrido todos los objetos, obteniendo un resultado que es solicitado por el cliente mediante el método getResults (el resultado es opcional ya que existen operaciones que no arrojan resultados).