Clase 12: Patrones de Diseño

Rodrigo Arturo Saffie Kattan

Pontificia Universidad Católica de Chile

rasaffie@ing.puc.cl

15 de septiembre de 2016

Contenidos

State

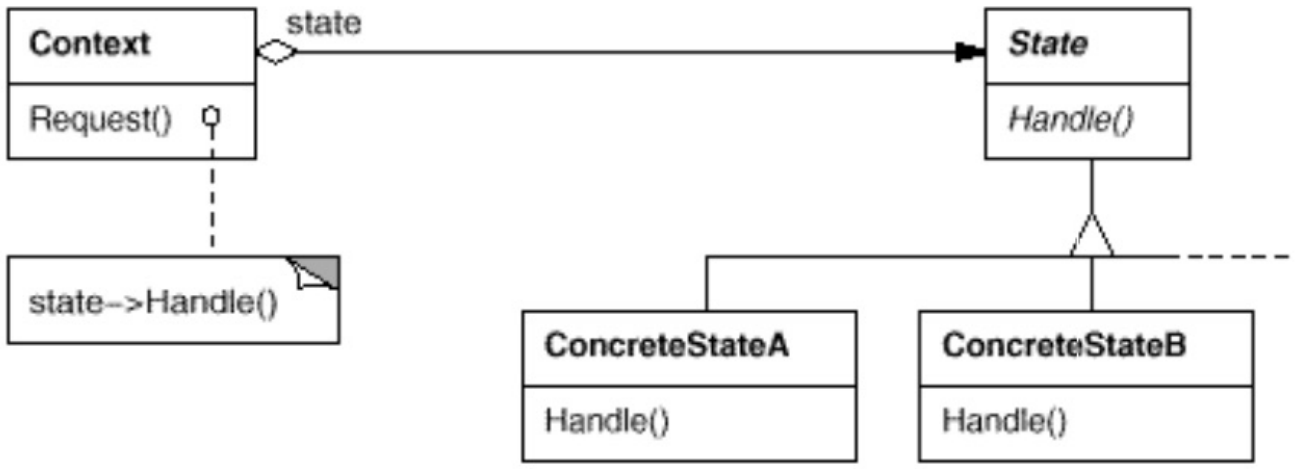
Permite a un objeto cambiar su comportamiento cuando su estado interno cambia. Parecerá que el objeto cambió de clase.

¿Cuándo se utiliza?

- El comportamiento de un objeto depende de su estado, y debe cambiar su comportamiento en tiempo de ejecución dependiendo de su estado.
- Las operaciones tienen gran cantidad de condicionales que dependen del estado del objeto. Estas condiciones son representadas por constantes, y/o se reutilizan en distintas operaciones. *State* permite encapsular cada una de las alternativas del condicional en una clase separada. Esto permite tratar el estado del objeto como un objeto en si, que puede variar independientemente.

State

Permite a un objeto cambiar su comportamiento cuando su estado interno cambia. Parecerá que el objeto cambió de clase.



Ejemplo : State

Strategy

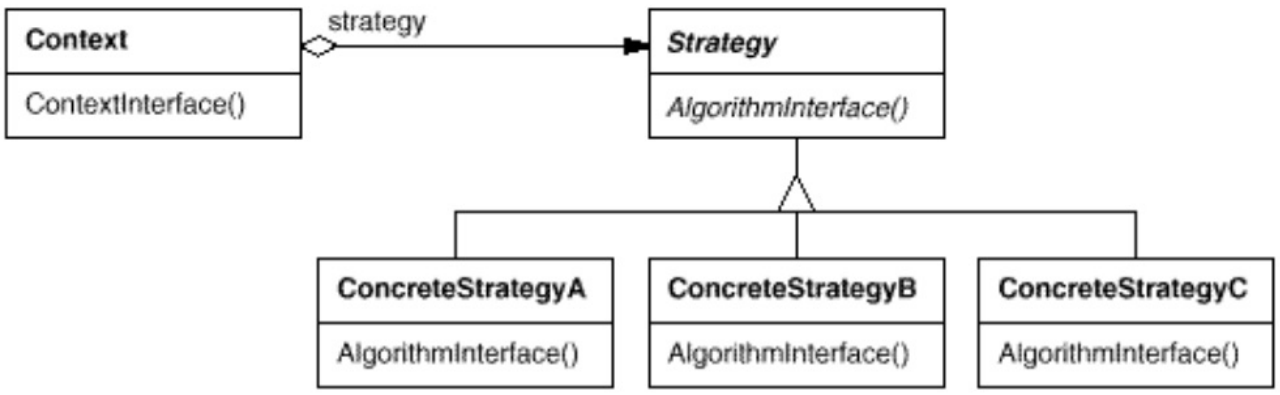
Define una familia de algoritmos, encapsula cada uno, y los hace intercambiables. *Strategy* permite al algoritmo cambiar independientemente de los clientes que lo utilizan.

¿Cuándo se utiliza?

- Varias clases relacionadas se diferencian solamente por su comportamiento. *Strategy* permite cambiar la configuración de una clase a través de la definición de su comportamiento.
- Se necesita diferenciar las variantes de un algoritmo.
- Para ocultar estructuras complejas utilizadas solamente por un algoritmo.
- Una clase define muchos comportamientos, que aparecen como condicionales en sus operaciones. En vez de reutilizar condicionales, se pueden encapsular alternativas bajo una clase *Strategy*.

Strategy

Define una familia de algoritmos, encapsula cada uno, y los hace intercambiables. *Strategy* permite al algoritmo cambiar independientemente de los clientes que lo utilizan.



Ejemplo : Strategy

Template Method

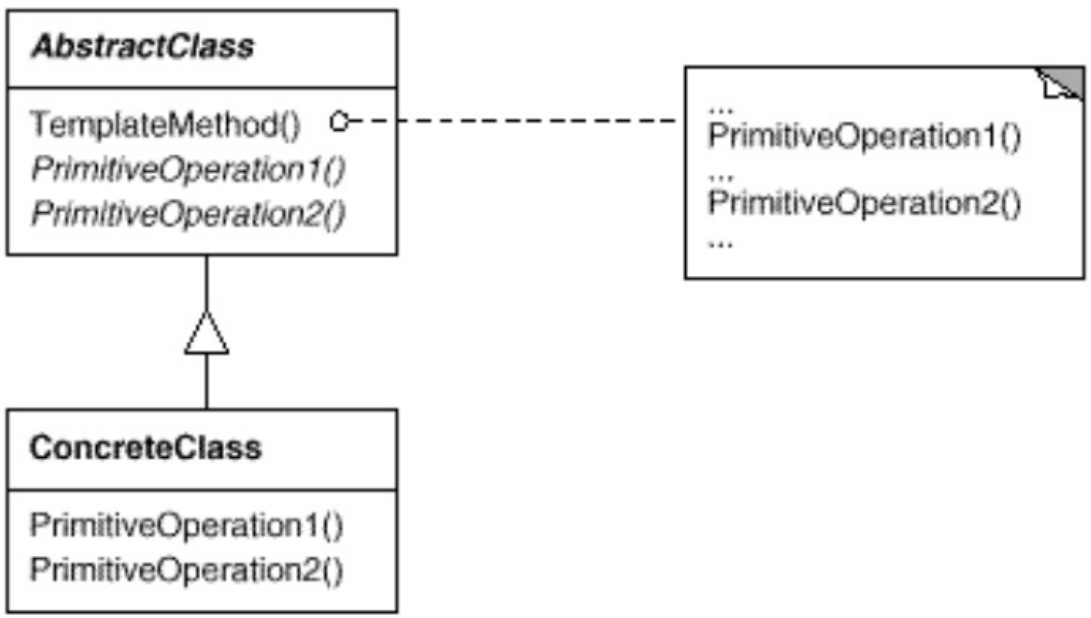
Define la estructura de un algoritmo, delegando algunos pasos a sub-clases. *Template Method* permite que las sub-clases redefinan algunos pasos del algoritmo, sin cambiar la estructura de este.

¿Cuándo se utiliza?

- Para implementar una sola vez la parte invariante de un algoritmo, y permitir que algunos pasos cambien
- Para agrupar compartimiento común, y así evitar código duplicado

Template Method

Define la estructura de un algoritmo, delegando algunos pasos a sub-clases. *Template Method* permite que las sub-clases redefinan algunos pasos del algoritmo, sin cambiar la estructura de este.



Ejemplo : Template Method

Visitor

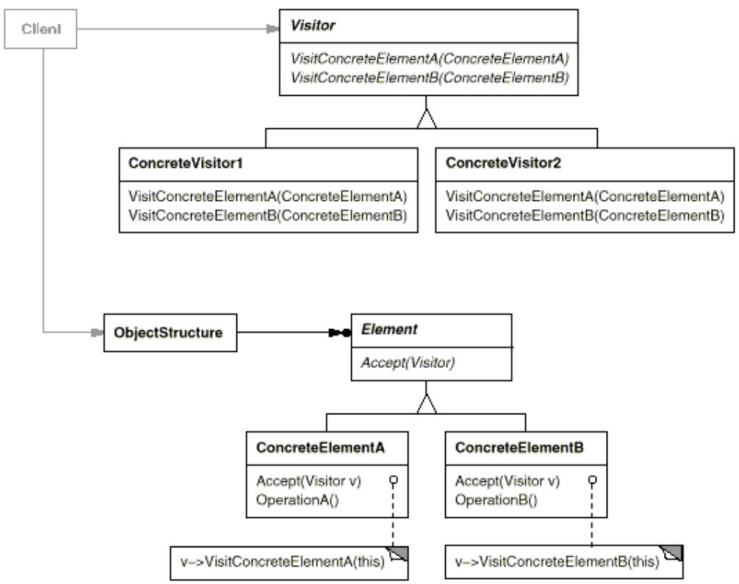
Representa una operación que sea ejecutada en los elementos que componen la estructura de un objeto. *Visitor* permite definir una nueva operación sin cambiar las clases de los elementos en los cuales se aplica.

¿Cuándo se utiliza?

- La estructura de un objeto contiene diversas clases de objetos con distintas interfaces, y se desea ejecutar operaciones en esos objetos independientemente de las clases en concreto.
- Muchas operaciones distintas y sin relación se aplican en los objetos de una estructura, y se quiere mantener estas clases simples, sin la declaración de estas operaciones

Visitor

Representa una operación que sea ejecutada en los elementos que componen la estructura de un objeto. *Visitor* permite definir una nueva operación sin cambiar las clases de los elementos en los cuales se aplica.



Ejemplo : Visitor

Referencias

Pressman, R. S. (2009)

Software Engineering: A Practitioner's Approach

7th ed., *McGraw-Hill Education*

Gamma, E., Helm. R., Johnson R., Vlissides, J. (1994)

Design Patterns: Elements of Reusable Object-Oriented Software

1st ed., *Addison-Wesley Professional*

dofactory.com

http://www.dofactory.com/net/design-patterns

Fin