Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Clase 5
Patrones de Diseño

IIC2113 – Diseño Detallado de Software

Rodrigo Saffie

rasaffie@uc.cl

27 de septiembre de 2018

Tarea 1

- Objetivos
  - Justificar decisiones de diseño en base a supuestos razonables.
  - Aplicar sintaxis *UML 2.0* para definir un sistema de *software* a través de distintos tipos de diagramas.
  - Plasmar el diseño de un software en un programa funcional.

I1

- Contenidos
  - Principios fundamentales del diseño
  - Modelo 4+1 / Diagramas UML
  - Principios SOLID
  - Patrones de Diseño

Encuesta Docente

- 5 respuestas de 69 alumnos
- Vamos rápido con la materia, pero las actividades son útiles para aprender
- Falta *feedback* de las actividades
- No es clara la utilidad de los patrones de diseño

Actividad 4

- Patrones de diseño
  - *Abstract Factory*
  - *Builder*
  - *Factory Method*
  - *Prototype*
  - *Singleton*
  - *Command*
  - *Iterator*
  - *Memento*
  - *Observer*

Patrones de diseño

De comportamiento:

- Se centran en las interacciones y responsabilidades entre objetos

Patrones de comportamiento

Chain of Responsibility

Evita el acoplamiento entre el objeto que envía una solicitud y el objeto que la recibe, al dar la oportunidad de que más de un objeto procese la solicitud. Encadena los objetos solicitados y delega la solicitud a lo largo de la cadena hasta que un objeto se encargue de ella.

Patrones de comportamiento

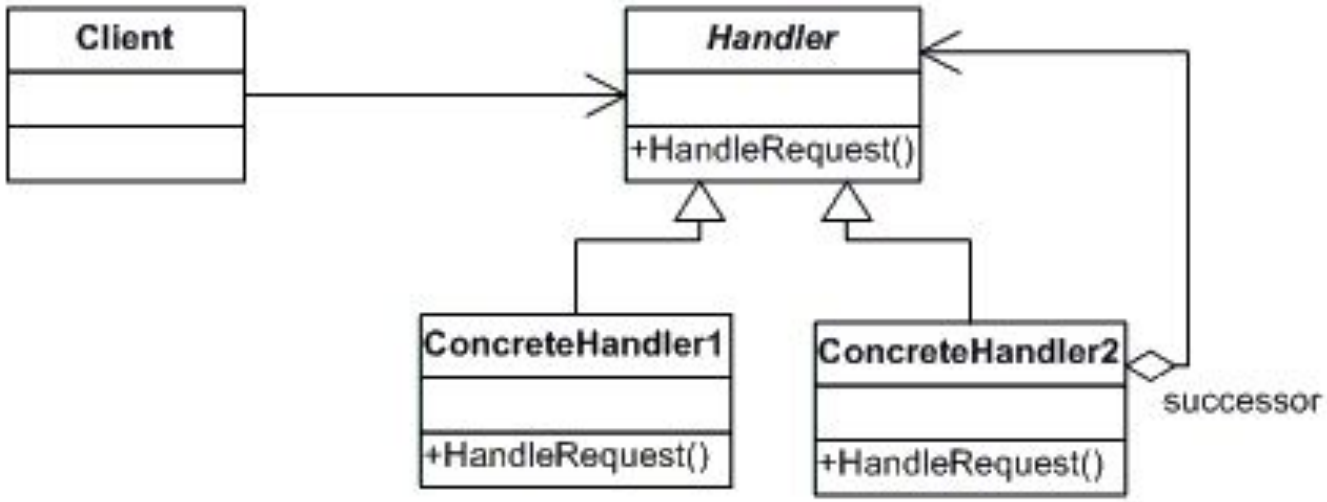
Chain of Responsibility

¿Cuándo se utiliza?

- Más de un objeto puede encargarse de una solicitud, y el que se hará cargo es desconocido.
- Se quiere generar una solicitud a uno de varios objetos sin especificar el receptor.
- El conjunto de objetos que puede hacerse cargo de la solicitud se define dinámicamente.

Patrones de comportamiento

Chain of Responsibility



Ejemplo

Patrones de comportamiento

Interpreter

Dado un lenguaje, define una representación de su gramática junto con un intérprete que utiliza la representación para interpretar frases en el idioma.

Patrones de comportamiento

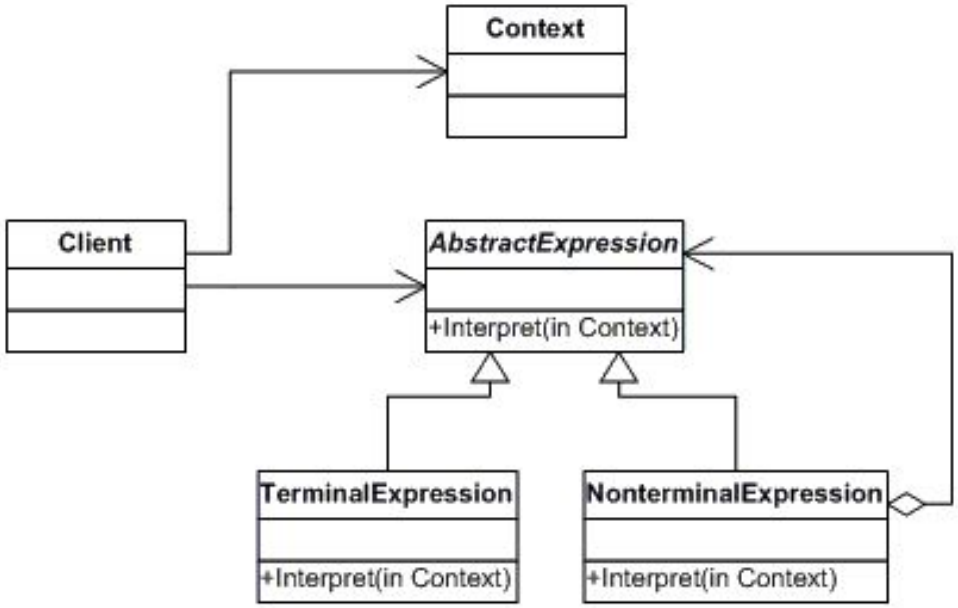
Interpreter

¿Cuándo se utiliza?

- La gramática del lenguaje es simple
- La eficiencia no es un requisito principal

Patrones de comportamiento

Interpreter



Ejemplo

Patrones de comportamiento

Mediator

Define un objeto que encapsula como un conjunto de objetos interactúan. Este patrón promueve el bajo acoplamiento al evitar que los objetos se referencien entre ellos explícitamente, y permite variar sus interacciones independientemente.

Patrones de comportamiento

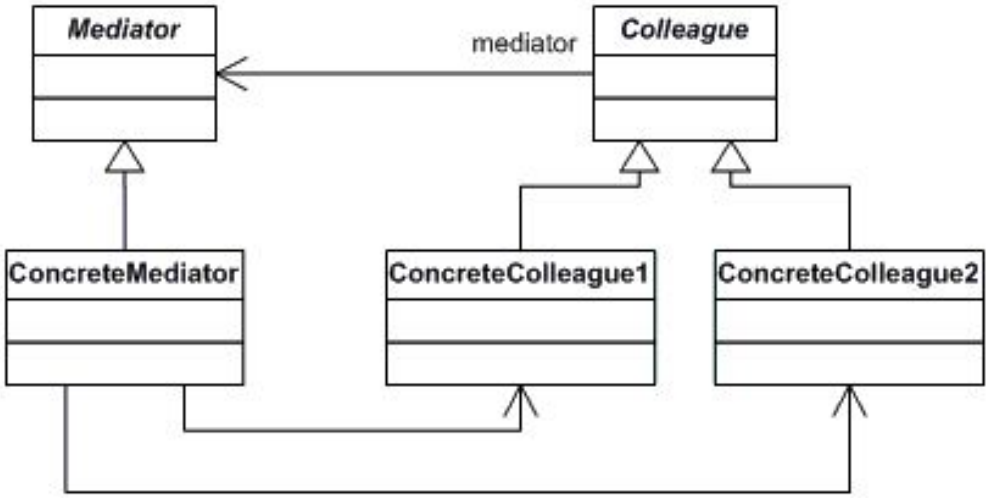
Mediator

¿Cuándo se utiliza?

- Un conjunto de objetos se comunican de una forma compleja, pero bien definida.
- Un comportamiento distribuido en múltiples clases debería ser configurable sin generar muchas sub-clases.

Patrones de comportamiento

Mediator



Ejemplo

Patrones de comportamiento

State

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

Patrones de comportamiento

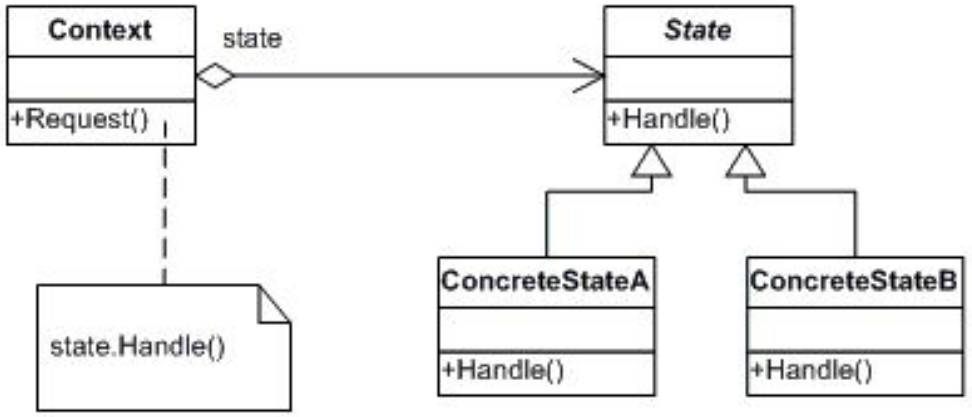
State

¿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 sí, que puede variar independientemente.

Patrones de comportamiento

State



Ejemplo

Patrones de comportamiento

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.

Patrones de comportamiento

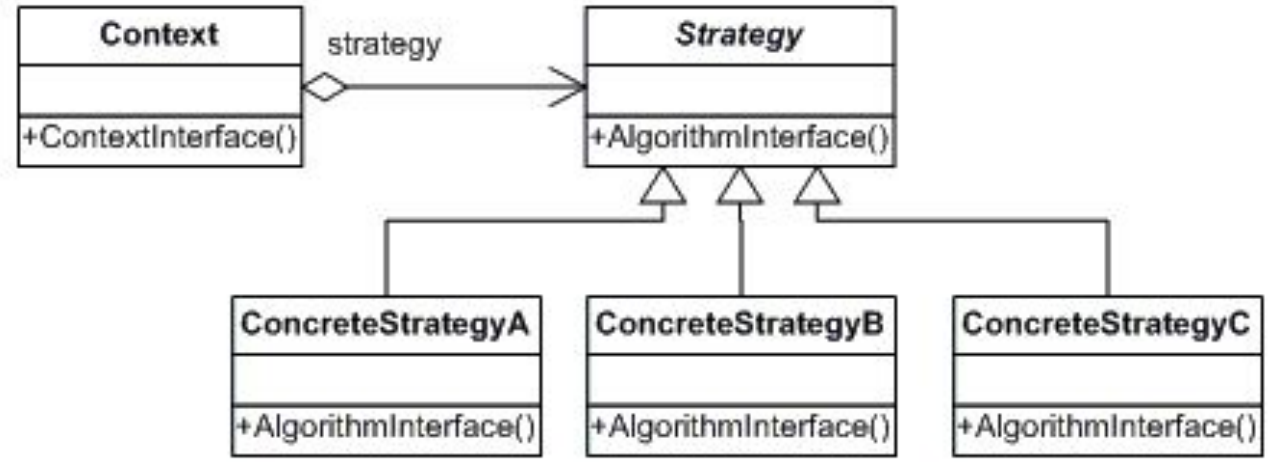
Strategy

¿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 diferencias 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*.

Patrones de comportamiento

Strategy



Ejemplo

Patrones de comportamiento

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.

Patrones de comportamiento

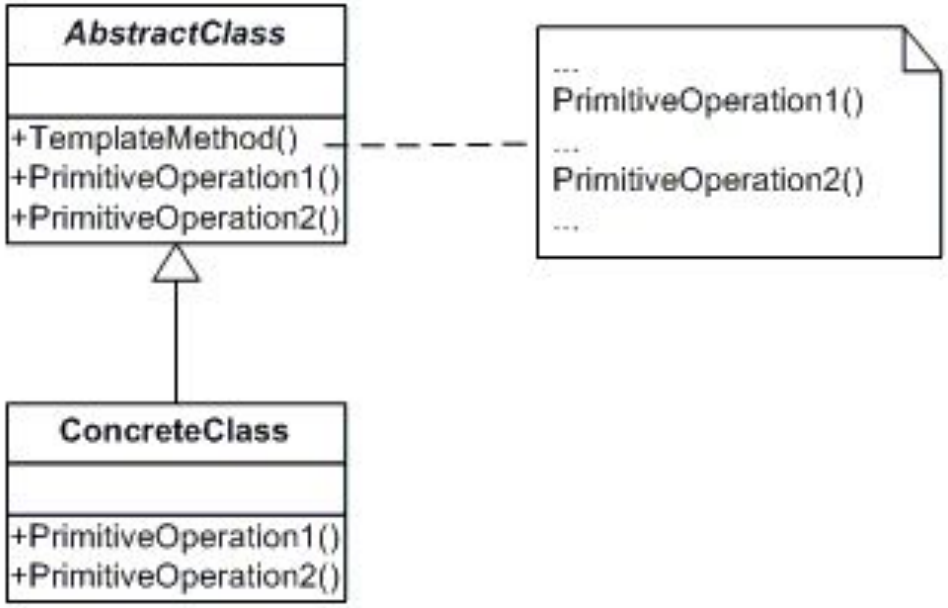
Template Method

¿Cuándo se utiliza?

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

Patrones de comportamiento

Template Method



Ejemplo

Patrones de comportamiento

Visitor

Representa una operación para 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.

Patrones de comportamiento

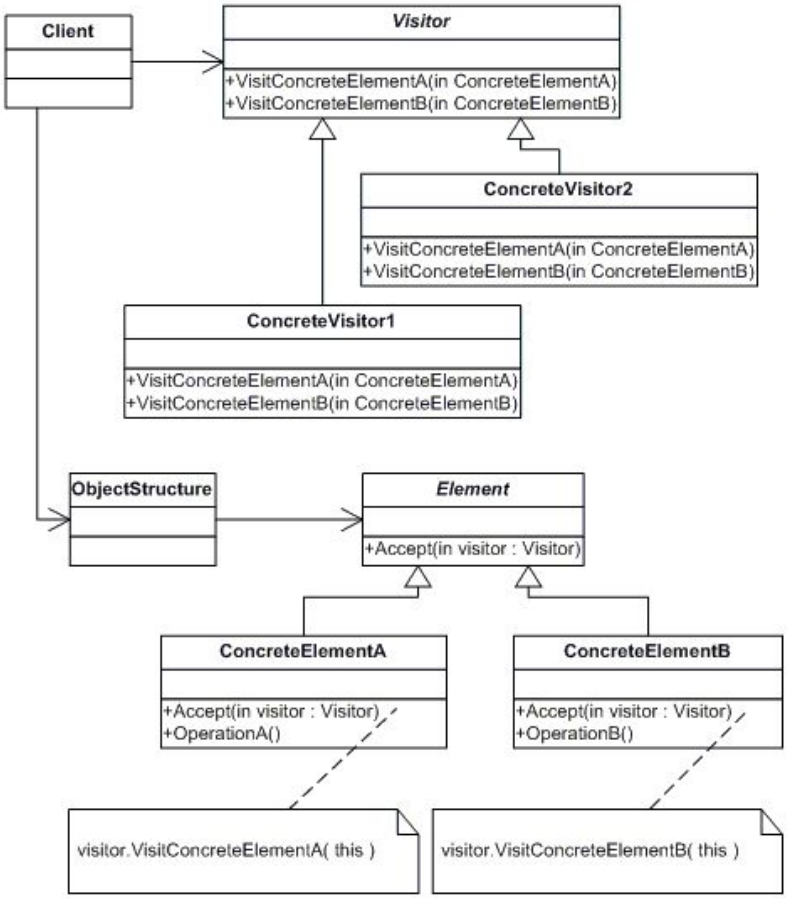
Visitor

¿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

Patrones de comportamiento

Visitor



Ejemplo

Actividad 5

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Clase 5
Patrones de Diseño

IIC2113 – Diseño Detallado de Software

Rodrigo Saffie

rasaffie@uc.cl

27 de septiembre de 2018