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

Clase 4
Patrones de Diseño

IIC2113 – Diseño Detallado de Software

Rodrigo Saffie

rasaffie@uc.cl

7 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.

Actividad 3

Patrones de diseño estructurales: se centran en cómo los objetos se organizan e integran en un sistema

- *Façade*
- *Adapter*
- *Composite*
- *Decorator*
- *Bridge*
- *Proxy*
- *Flyweight*

Patrones de diseño

Creacionales:

- Se centran en la creación, composición y representación de los objetos

Patrones creacionales

Abstract Factory

Provee una interfaz para la creación de familias de objetos relacionados o dependientes, sin especificar concretamente sus clases.

Patrones creacionales

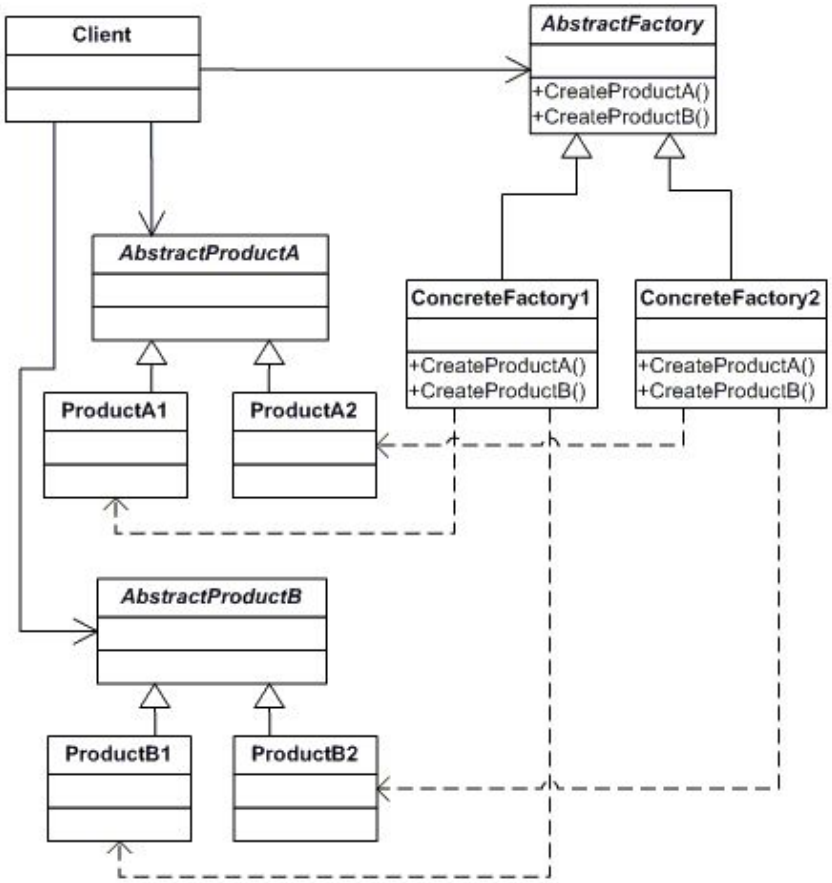
Abstract Factory

¿Cuándo se utiliza?

- Un sistema debe ser independiente de los productos que crea, compone y representa
- Un sistema debe ser configurado con una familia de productos entre múltiples disponibles
- Una familia de productos relacionados está diseñada para ser utilizada en conjunto, y se debe forzar esta regla
- Se desea proveer una librería de productos, pero ocultar su implementación

Patrones creacionales

Abstract Factory



[Ejemplo]

Patrones creacionales

Builder

Separa la construcción de un objeto complejo de su representación, para que así el mismo proceso de construcción pueda crear diferentes representaciones.

Patrones creacionales

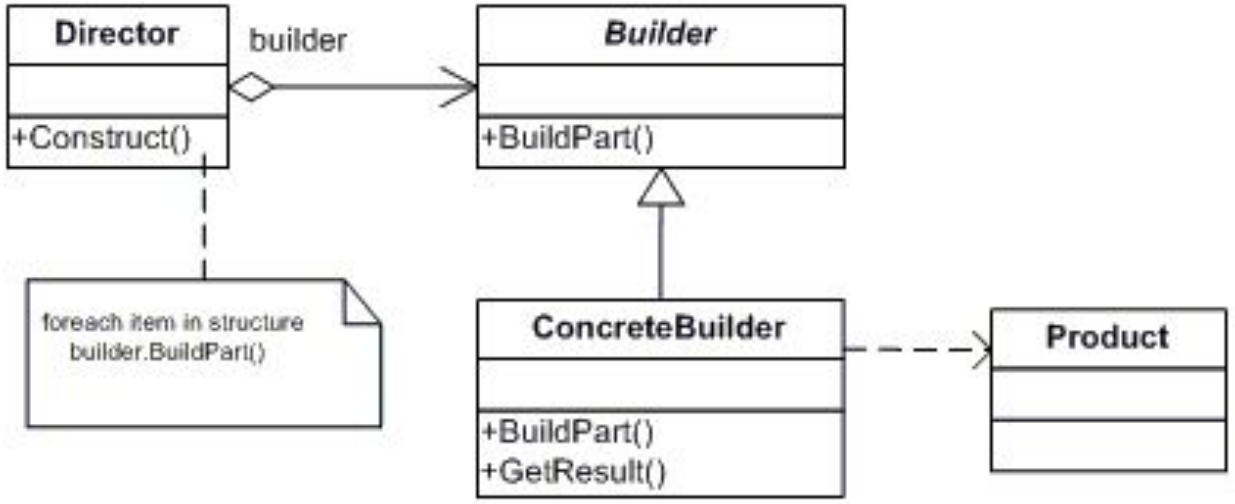
Builder

¿Cuándo se utiliza?

- El algoritmo para crear objetos complejos debería ser independiente de las partes que componen el objeto, y de cómo se construyen
- El proceso de construcción debe permitir diferentes representaciones del objeto que se construye

Patrones creacionales

Builder



Ejemplo

Patrones creacionales

Factory Method

Define una interfaz para la creación de un objeto, pero delega a las subclases que decidan qué clase instanciar.

Patrones creacionales

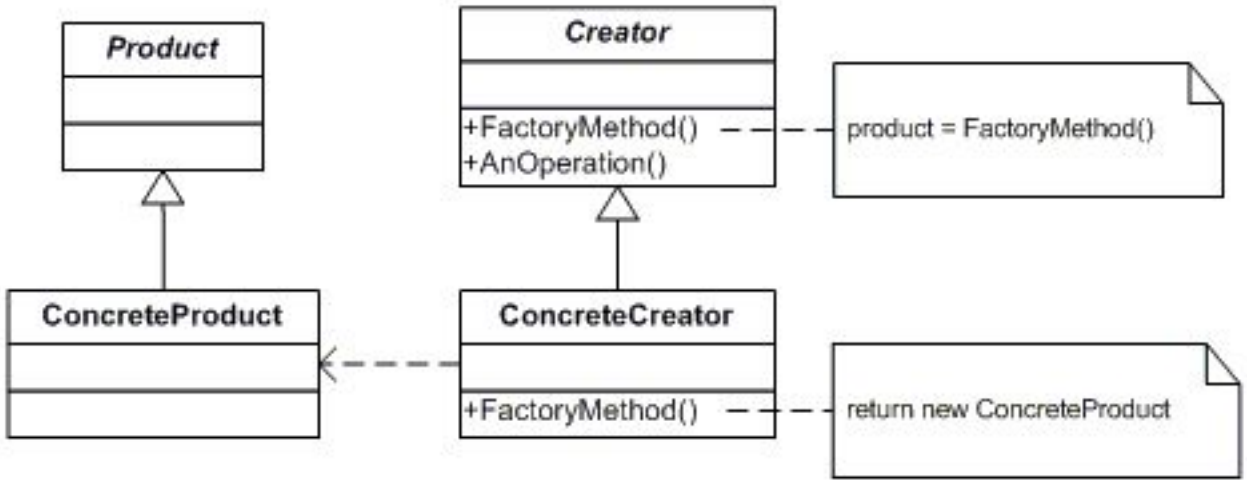
Factory Method

¿Cuándo se utiliza?

- Una clase no puede anticipar las clases de los objetos que debe crear
- Una clase necesita permitir a sus subclases especificar la creación de objetos
- Una clase delega la responsabilidad de crear objetos a sus subclases, para así encapsular lógica

Patrones creacionales

Factory Method



Ejemplo

Patrones creacionales

Prototype

Especifica los tipos de objetos a crear utilizando prototipos, y crea objetos nuevos a través de copias de estos prototipos.

Patrones creacionales

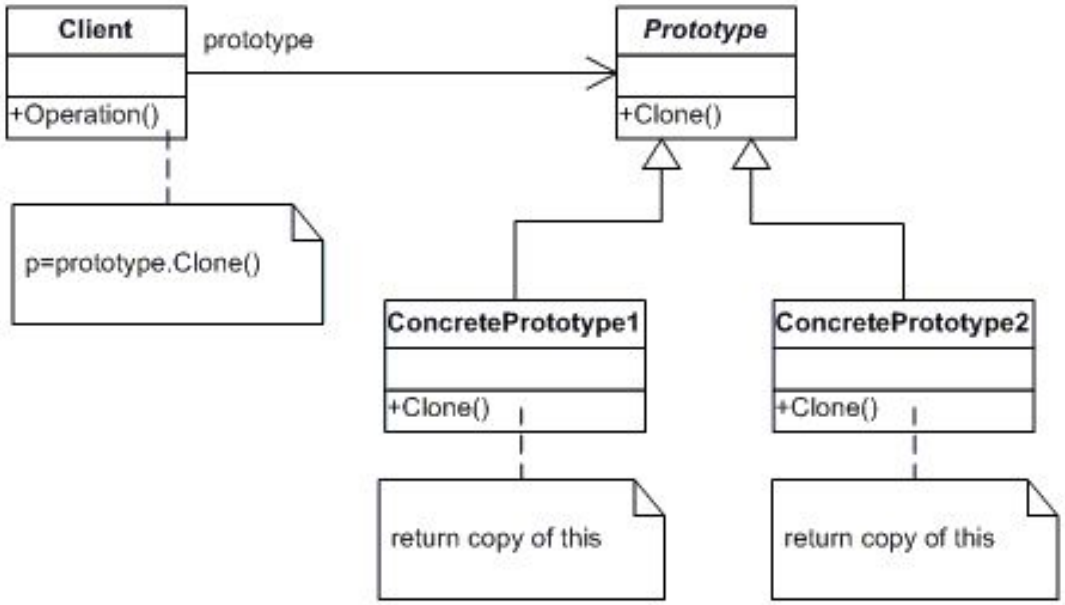
Prototype

¿Cuándo se utiliza?

- Las clases a crear se especifican en tiempo de ejecución
- Los objetos a crear son similares, y se pueden desprender de un objeto existente
- Instancias de una clase tienen una cantidad limitada de estados. Puede ser más conveniente instanciar una sola vez estos estados, para luego entregar copias

Patrones creacionales

Prototype



Ejemplo

Patrones creacionales

Singleton

Garantiza que una clase tenga solamente una instancia, y provee un acceso global a la instancia.

Patrones creacionales

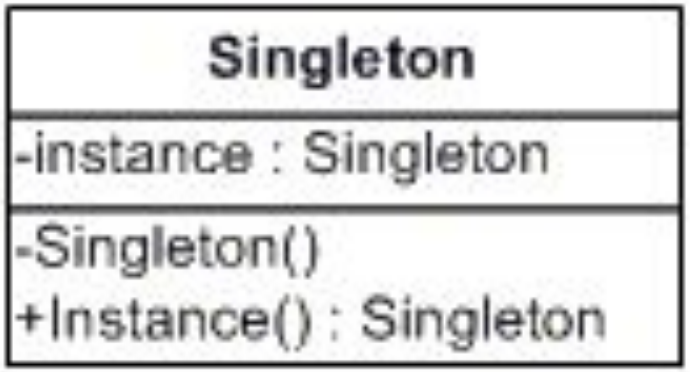
Singleton

¿Cuándo se utiliza?

- Debe haber exactamente una sola instancia de una clase, y puede ser accedida por distintos clientes

Patrones creacionales

Singleton



Ejemplo

Patrones de diseño

De comportamiento:

- Se centran en las interacciones y responsabilidades entre objetos

Patrones de comportamiento

Command

Encapsula una solicitud como un objeto, para así permitir la parametrización de clientes con solicitudes distintas, encolar o registrar solicitudes, y soportar operaciones que se pueden deshacer.

Patrones de comportamiento

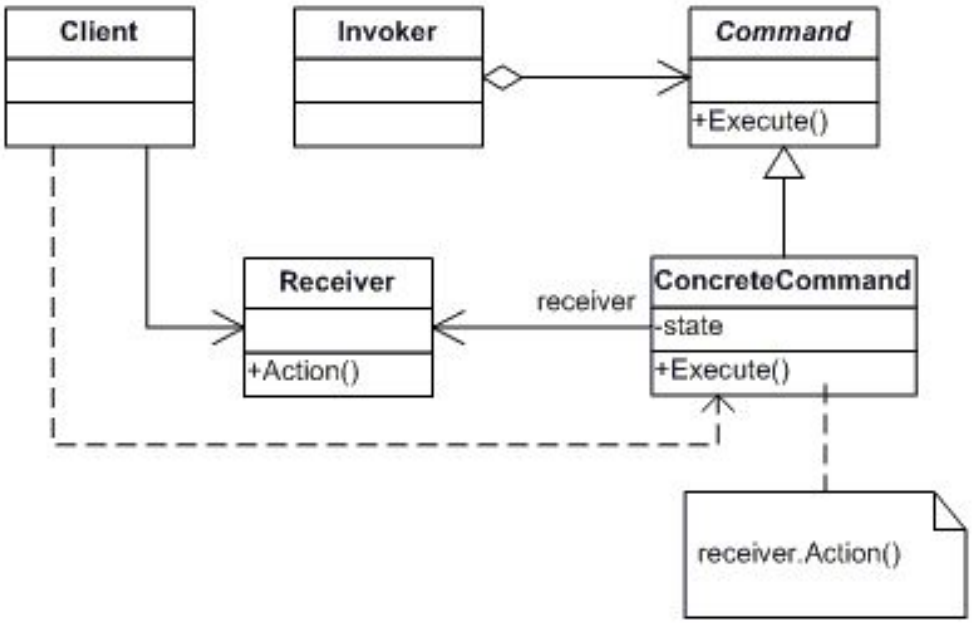
Command

¿Cuándo se utiliza?

- Se necesita parametrizar objetos con una acción a realizar. *Command* es un reemplazo para *callbacks* en OOP
- Especificar, encolar y ejecutar solicitudes en momentos diferentes
- Soportar que la solicitud se deshaga
- Estructurar un sistema alrededor de operaciones complejas, formadas a partir de operaciones primitivas

Patrones de comportamiento

Command



Ejemplo

Patrones de comportamiento

Iterator

Provee una forma de acceder a los elementos de una agregación de objetos secuenciales sin exponer su representación subayacente.

Patrones de comportamiento

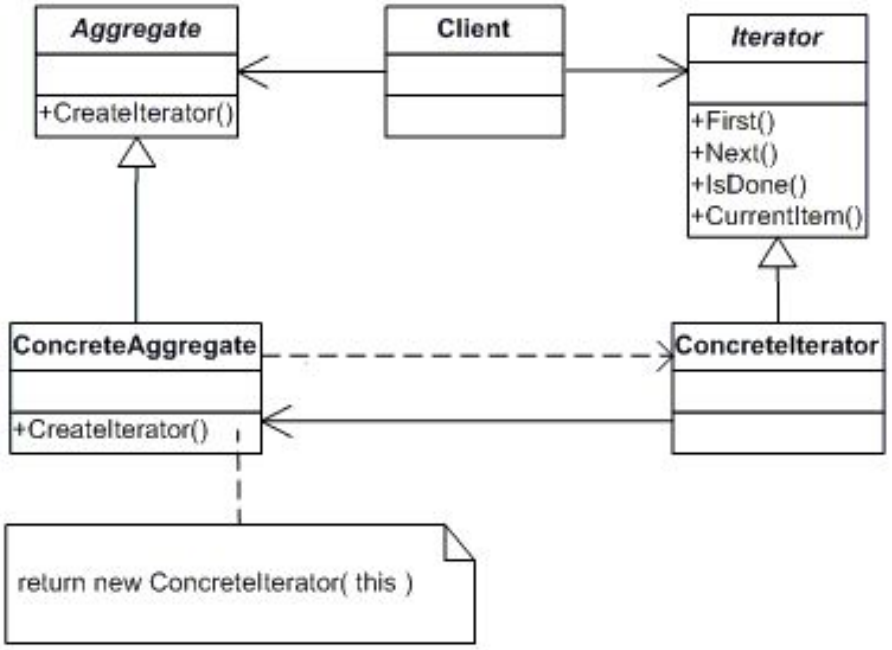
Iterator

¿Cuándo se utiliza?

- Para acceder al contenido de un objeto agregado sin exponer su representación interna
- Para soportar múltiples recorridos de objetos agregados
- Para proveer una interfaz uniforme para recorrer diferentes estructuras agregadas

Patrones de comportamiento

Iterator



Ejemplo

Patrones de comportamiento

Memento

Sin violar el principio de ocultamiento, captura y expone el estado interno de un objeto para que este pueda volver a ese estado en otro momento.

Patrones de comportamiento

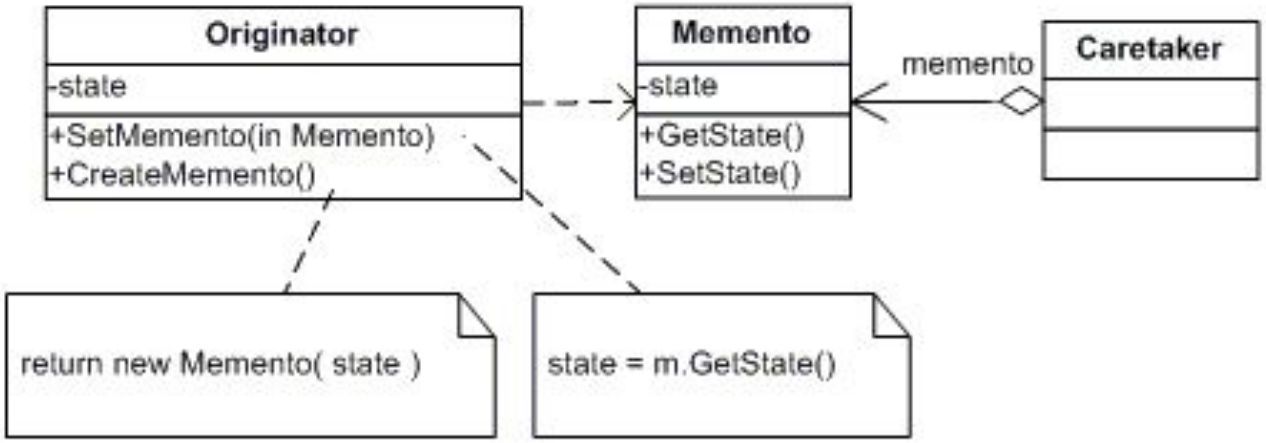
Memento

¿Cuándo se utiliza?

- Una “fotografía” de una parte del estado de un objeto debe ser grabada para poder restaurarla.
- Una interfaz directa para obtener el estado de un objeto expondría detalles de implementación, violando el principio de ocultamiento.

Patrones de comportamiento

Memento



Ejemplo

Patrones de comportamiento

Observer

Define una relación de dependencia entre uno y N objetos, para que cuando ese objeto cambie su estado, todos los N sean notificados y se actualicen automáticamente.

Patrones de comportamiento

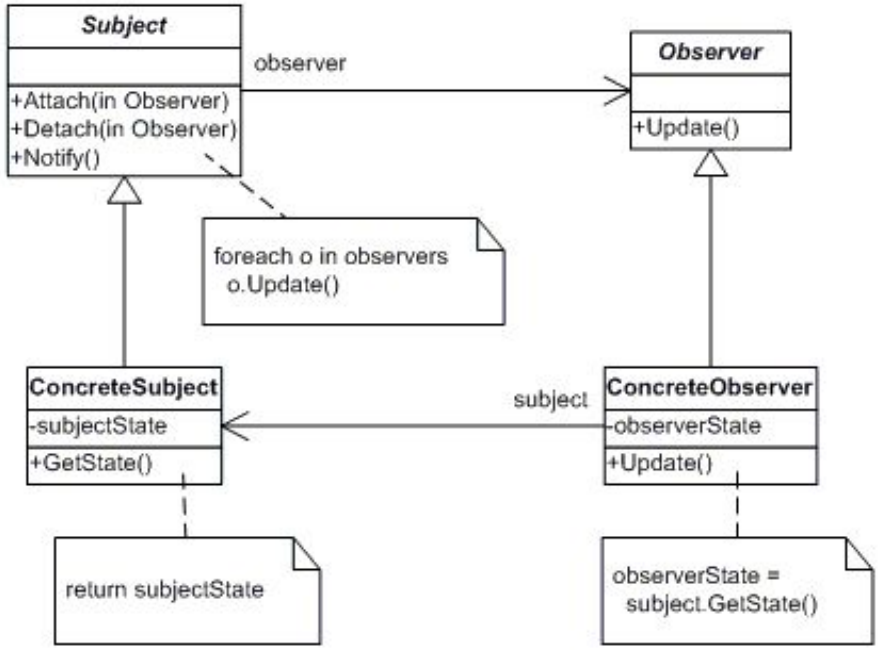
Observer

¿Cuándo se utiliza?

- Cuando una abstracción tiene dos aspectos, uno dependiente del otro. Encapsular estos aspectos en objetos separados permite variarlos y reutilizarlos independientemente.
- Cuando un cambio en un objeto genera cambios en otros, y no se sabe cuántos objetos deben cambiar.
- Cuando un objeto debería ser capaz de notificar a otros objetos sin realizar supuestos de quiénes son estos objetos. En otras palabras, se quiere reducir el acoplamiento entre objetos dependientes.

Patrones de comportamiento

Observer



Ejemplo

Actividad 4

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

Clase 4
Patrones de Diseño

IIC2113 – Diseño Detallado de Software

Rodrigo Saffie

rasaffie@uc.cl

7 de septiembre de 2018