



# Análisis y Diseño de Sistemas 2

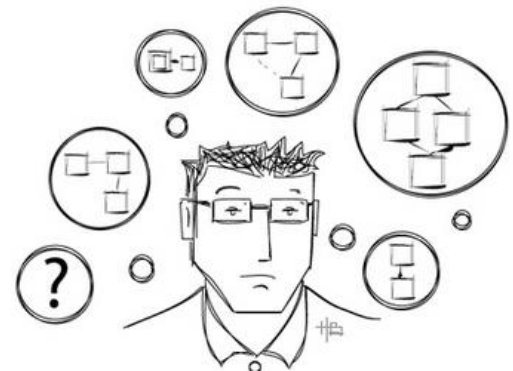
2015

Ing. Luis Alberto Arias Solórzano

Unidad 3

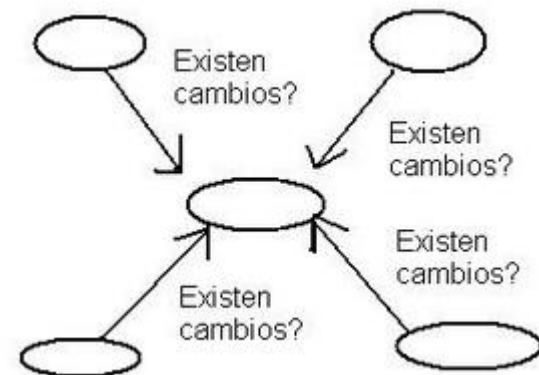
# Patrones de Diseño

- ▶ En la ingeniería de software, un patrón de diseño es una solución general reutilizable a un problema que ocurre comúnmente en un contexto determinado en el diseño de software.
- ▶ Un patrón de diseño no es un diseño o solución final que se puede transformar directamente en código fuente. Es una descripción o una plantilla para la forma de resolver un problema que se puede utilizar en muchas situaciones diferentes.
- ▶ Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Se formalizaron como mejores prácticas que el programador puede utilizar para resolver los problemas comunes en el diseño de una aplicación o sistema.



# Patrones de Diseño


- ▶ Los Patrones de diseño orientado a objetos suelen mostrar las relaciones e interacciones entre clases u objetos, sin especificar las clases u objetos que están involucrados en la solución final. Los patrones que implican la orientación a objetos buscan añadir capas de abstracción. Cuando se abstrae algo, se están aislando detalles concretos, y una de las razones de mayor peso para hacerlo es separar las cosas que cambian de las cosas que no.
- ▶ Los patrones de diseño se agruparon inicialmente en las categorías: **Patrones de creación** (creacionales), **Patrones de Estructura** (estructurales) y **Patrones de Comportamiento**; estos se describen utilizando los conceptos de delegación, agregación y la consulta.



# Patrones de Creación

Los patrones creacionales son los que crean objetos y aíslan sus detalles, en lugar de tener que crear instancias directamente objetos. Esto le da a su programa más flexibilidad para decidir qué objetos se deben crear para un caso en específico, de forma que su código no dependa de los tipos de objeto que hay y por lo tanto, no tenga que cambiarlo cuando añada un nuevo tipo de objeto.

Ejemplos:

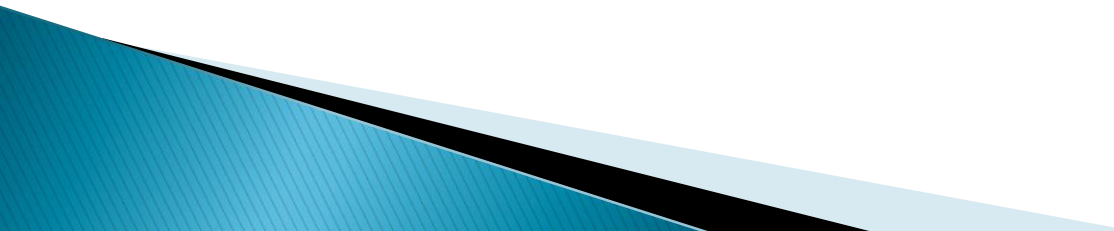
- ▶ **Abstract Factory:** grupos de objetos que tienen un tema en común.
  - ▶ **Builder:** Construye objetos complejos separando su construcción y representación.
  - ▶ **Factory Method:** crea objetos sin especificar exactamente la clase a crear.
  - ▶ **Prototype:** crea objetos clonando un objeto existente.
  - ▶ **Singleton:** restringe la creación de objetos de una clase a una sola instancia.
- 

# Patrones de Creación – Singleton

El patrón singleton es un patrón de diseño que restringe la creación de instancias de una clase a un solo objeto. Esto es útil cuando se necesita exactamente de un objeto para coordinar las acciones de todo el sistema.

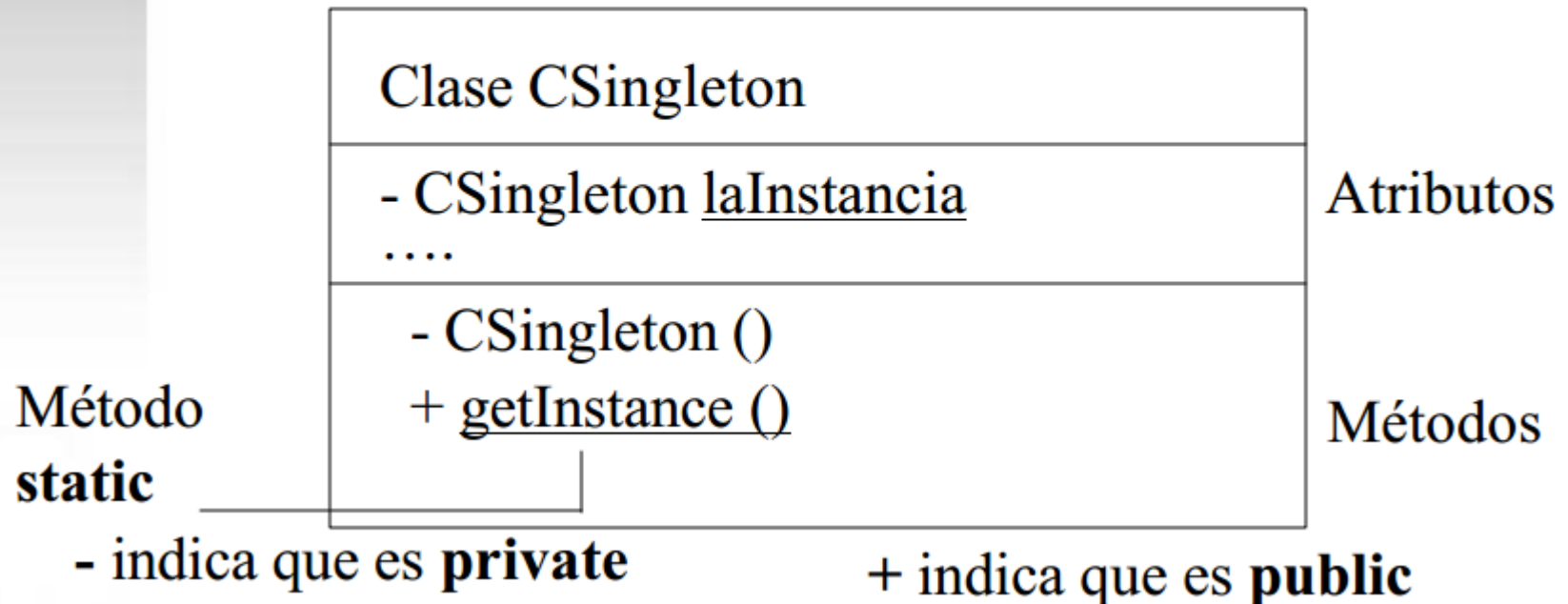
El concepto es a veces generalizado a sistemas que operan de manera más eficiente cuando sólo existe un objeto, o que restringen la creación de instancias a un cierto número de objetos. El término viene del concepto matemático de un singleton.

Hay críticas al uso del patrón singleton, ya que algunos consideran que es un anti-patrón, pues se considera que si es usado en exceso, introduce restricciones innecesarias en situaciones en las que en realidad no se requiere una única instancia de una clase, y se introducen estados globales en una aplicación.



# Patrones de Creación – Singleton

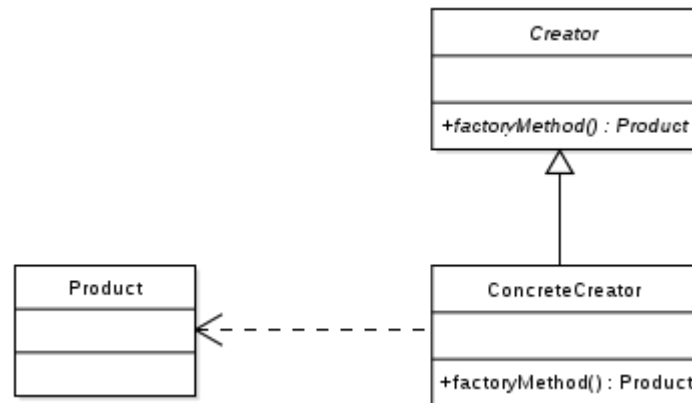
- El constructor de la clase DEBE SER PRIVADO
- Se proporciona un método ESTÁTICO en la clase que devuelve LA ÚNICA INSTANCIA DE LA CLASE: `getInstance()`



# Patrones de Creación – Factory Method

El patrón Factory Method es un patrón creacional que utiliza métodos de fábrica para hacer frente al problema de la creación de objetos sin especificar la clase exacta de objeto que se va a crear.

Esto se hace mediante la creación de objetos a través de un método de fábrica, que o bien se especifica en una interfaz (clase abstracta) e implementado en las clases (clases concretas) en la ejecución; o bien se implementa en una clase base (opcionalmente como un método de plantilla), que puede ser anulada cuando se hereda en las clases derivadas y no por un constructor.

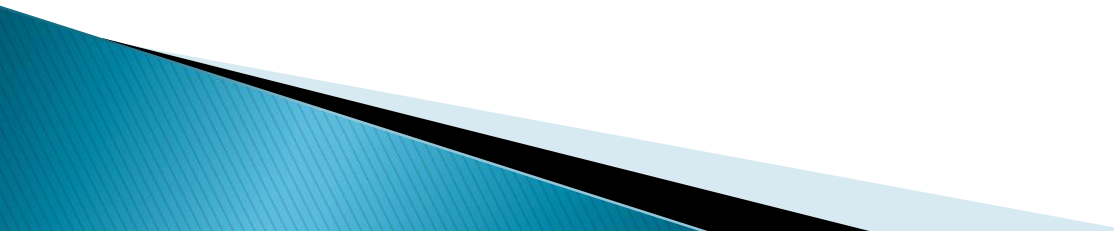




# Patrones de Estructura

Los patrones estructurales son los que se preocupan por la composición de las clases y objetos. Ellos utilizan la herencia para componer las interfaces y definir maneras para componer objetos para obtener nuevas funcionalidades. Afectando la manera en que los objetos se conectan unos con otros y para asegurar que los cambios en el sistema no requieren cambiar esas relaciones (conexiones).

Ejemplos:

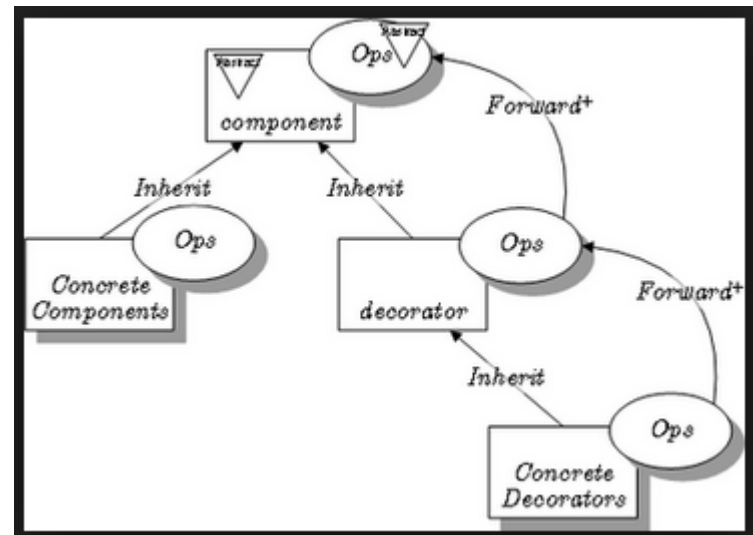
- ▶ Adapter: permite clases con interfaces incompatibles trabajar juntas envolviendo su propia interface con una clase existente.
  - ▶ Bridge: desacopla una abstracción de su implementación para que las dos puedan variar independientemente.
  - ▶ Composite: compone cero o mas objetos similares que pueden ser manipulados como solo un objeto.
- 



# Patrones de Estructura

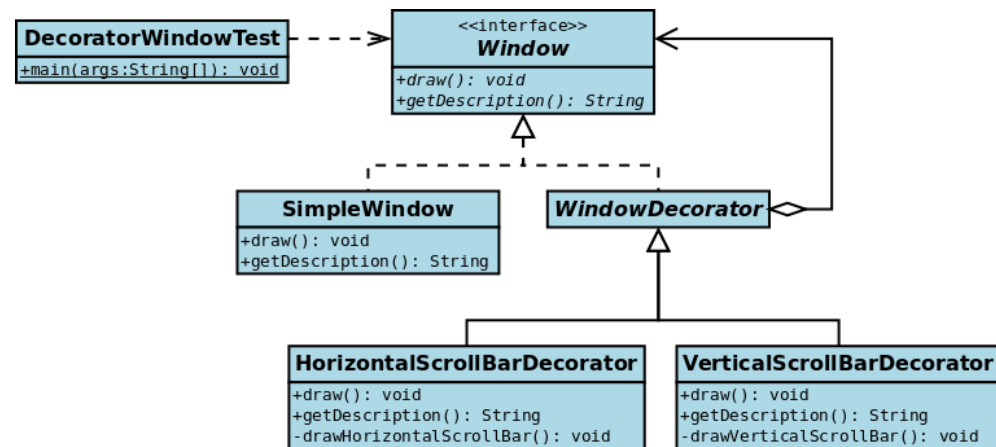
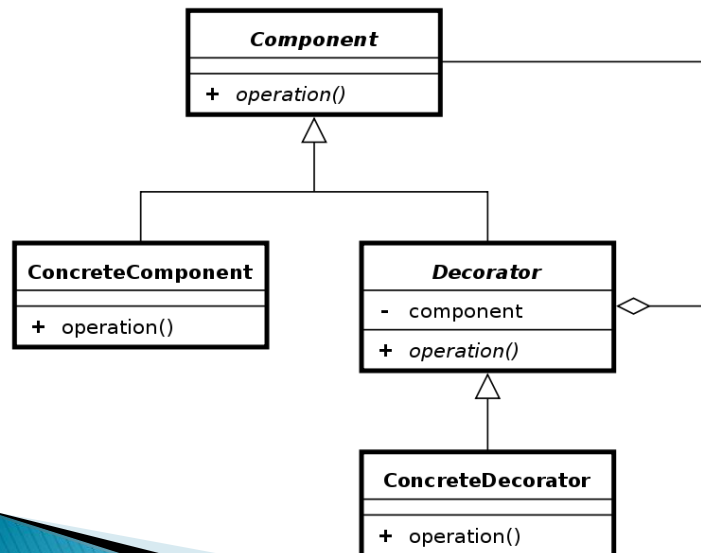
Ejemplos (continuación):

- ▶ Decorator: dinámicamente agrega o sobrescribe (overrides) comportamiento en un método existente del objeto.
- ▶ Facade: Fachada provee una interfaz simplificada o unificada para un largo trozo de codificación, o múltiples interfaces.
- ▶ Flyweight: reduce el costo de crear y manipular grandes grupos de objetos.
- ▶ Proxy: provee un lugar para colocar otro objeto que se encarga del control del acceso, reduciendo el costo y la complejidad.



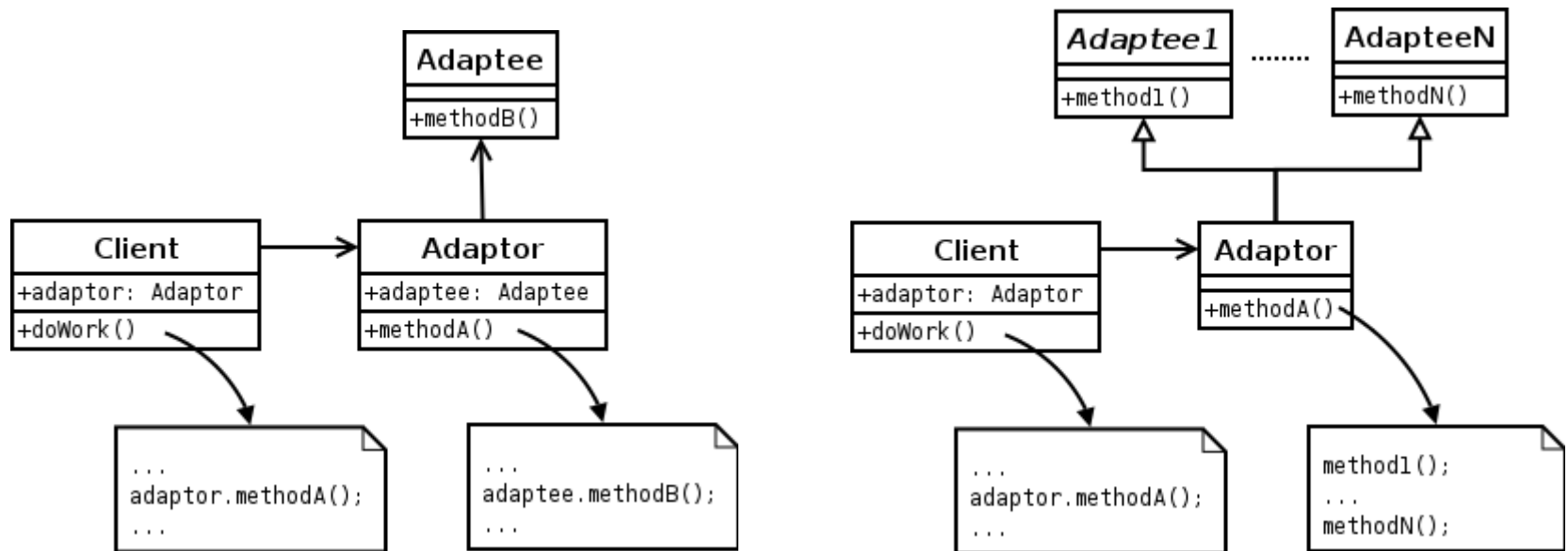
# Patrones de Estructura – Decorador

El patrón decorador (también conocida como envoltura, una nomenclatura alternativa compartido con el patrón adaptador) es un patrón de diseño que permite que al comportamiento que se añada un objeto individual, ya sea estática o dinámicamente, sin afectar el comportamiento de los otros objetos de la misma clase.

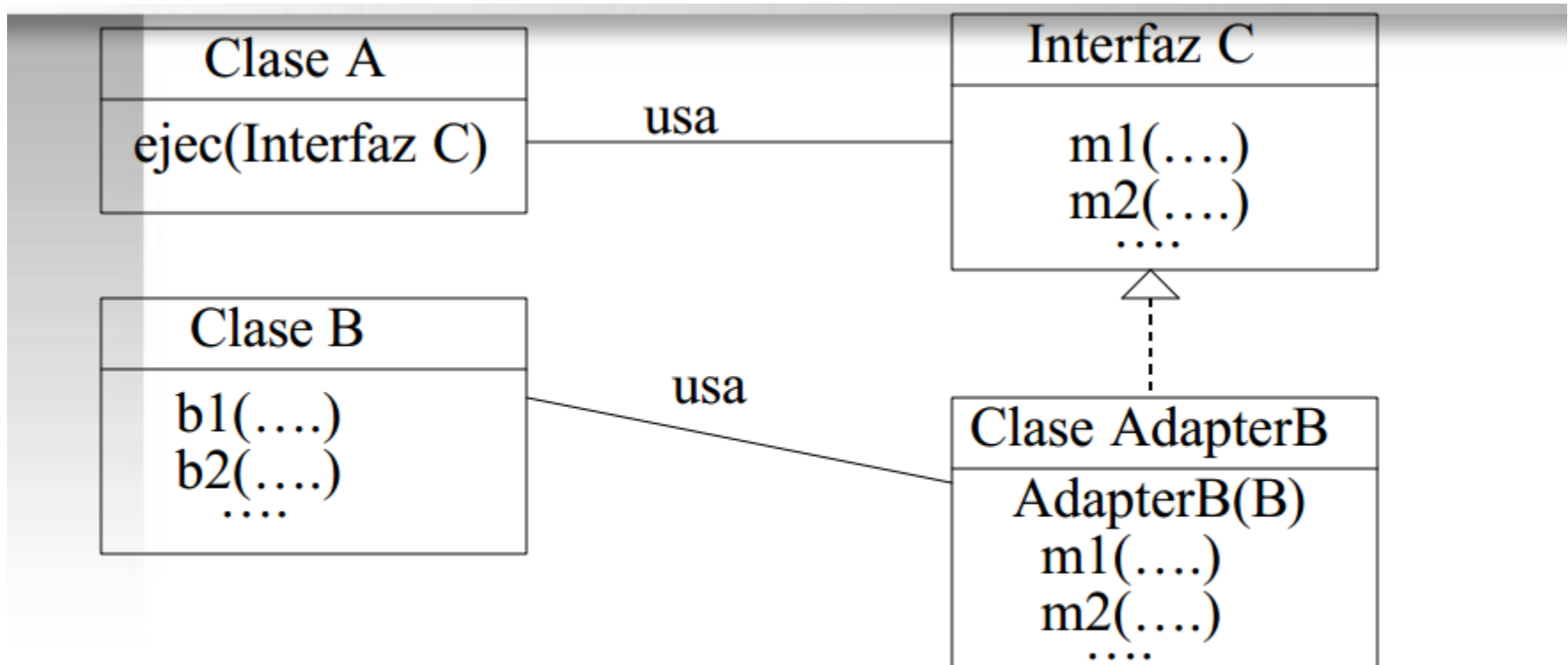


# Patrones de Estructura – Adaptador

El patrón adaptador es un patrón de diseño de software que permite que la interfaz de una clase existente sea utilizada por otra interfaz. A menudo se utiliza para hacer que las clases existentes trabajen con otros objetos sin necesidad de modificar su código fuente.



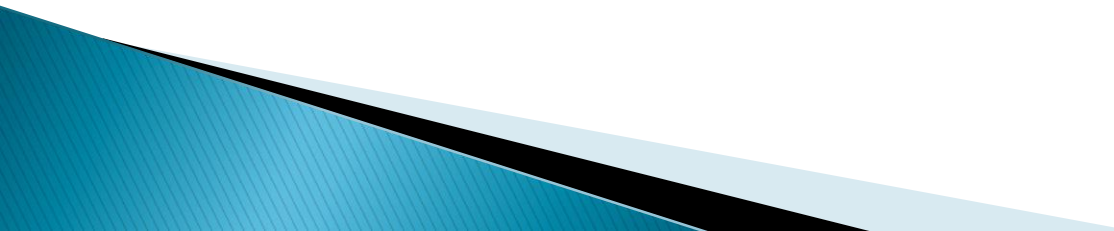
# Patrones de Estructura – Adaptador



**Solución:** construir una clase Adaptadora de B que implemente la interfaz C. Al implementarla, usa un objeto de B y sus métodos

# Patrones de Comportamiento

Los patrones de comportamiento o composición se definen como patrones de diseño software que ofrecen soluciones respecto a la interacción y responsabilidades entre clases y objetos, así como los algoritmos que encapsulan.

- ▶ **Chain of Responsibility:** Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
  - ▶ **Command:** Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
  - ▶ **Interpreter:** Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
  - ▶ **Iterator:** Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
- 

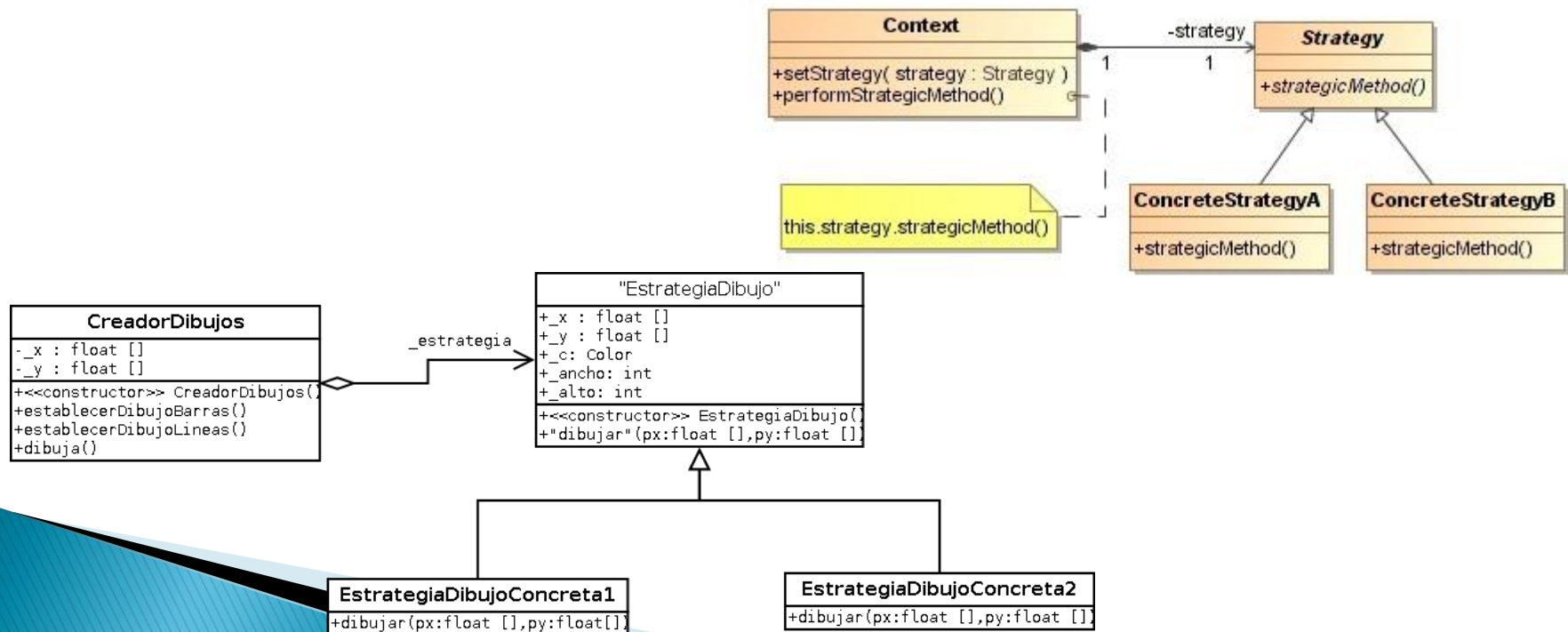
# Patrones de Comportamiento

Ejemplos (continuación):

- ▶ **Mediator:** Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
- ▶ **Memento:** Permite volver a estados anteriores del sistema.
- ▶ **Observer:** Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
- ▶ **State:** Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
- ▶ **Strategy:** Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
- ▶ **Template Method:** Define en una operación el esqueleto de un algoritmo, delegando en las subclasses algunos de sus pasos, esto permite que las subclasses redefinan ciertos pasos de un algoritmo sin cambiar su estructura.
- ▶ **Visitor:** Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

# Patrones de Comportamiento – Strategy

El patrón Strategy es un patrón de diseño para el desarrollo de software. Se clasifica como patrón de comportamiento porque determina como se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. El patrón estrategia permite mantener un conjunto de algoritmos de entre los cuales el objeto cliente puede elegir aquel que le conviene e intercambiarlo dinámicamente según sus necesidades.

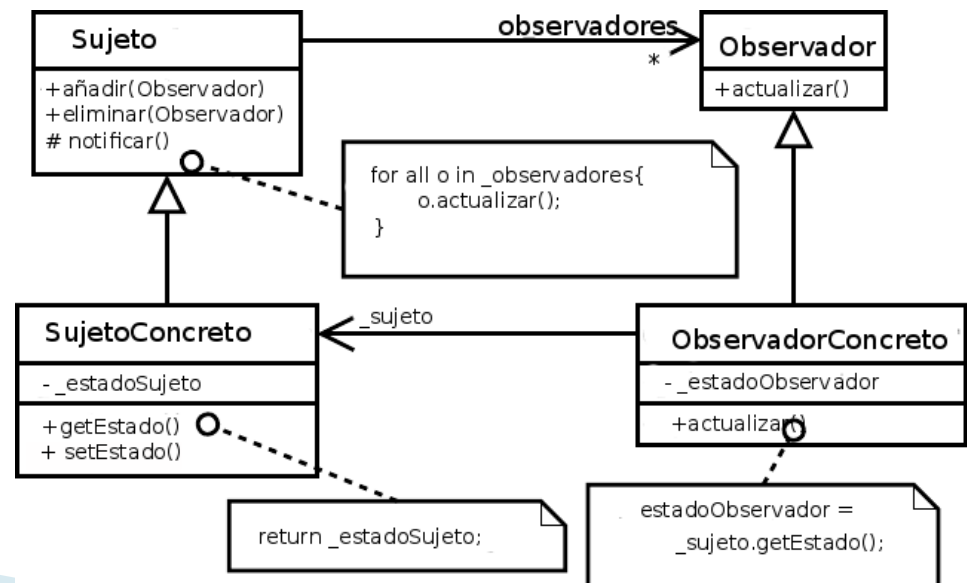




# Patrones de Comportamiento – Observador

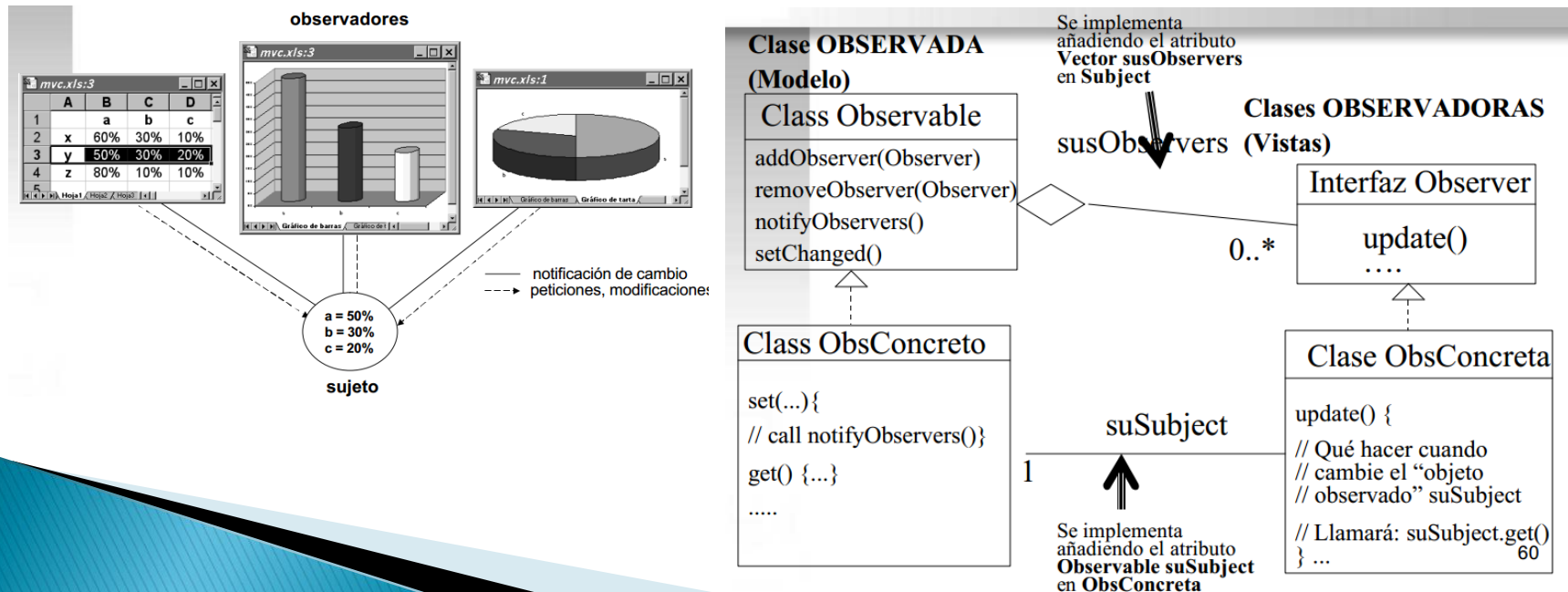
El patrón observador es un patrón de diseño que define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes.

La variación de la encapsulación es la base de muchos patrones de comportamiento, por lo que cuando un aspecto de un programa cambia frecuentemente, estos patrones definen un objeto que encapsula dicho aspecto. Los patrones definen una clase abstracta que describe la encapsulación del objeto.



# Patrones de Comportamiento – Observador

Este patrón también se conoce como el patrón de publicación–inscripción. El objeto inicial, que se le puede llamar “Sujeto”, contiene atributos mediante los cuales cualquier objeto “Observador” o vista se puede suscribir a él pasándole una referencia a sí mismo. El Sujeto mantiene así una lista de las referencias a sus observadores. Los observadores a su vez están obligados a implementar unos métodos determinados mediante los cuales el Sujeto es capaz de notificar a sus observadores suscritos los cambios que sufre para que todos ellos tengan la oportunidad de refrescar el contenido representado.



# Lecturas

1. <http://msdn.microsoft.com/es-es/library/bb972240.aspx>
  2. [http://en.wikipedia.org/wiki/Design\\_Patterns\\_\(book\)](http://en.wikipedia.org/wiki/Design_Patterns_(book))
  3. [http://en.wikipedia.org/wiki/Software\\_design\\_pattern](http://en.wikipedia.org/wiki/Software_design_pattern)
  4. [http://arco.esi.uclm.es/~david.villa/pensar\\_en\\_C++/vol2/C10.html](http://arco.esi.uclm.es/~david.villa/pensar_en_C++/vol2/C10.html)
- [http://es.wikipedia.org/wiki/Patr%C3%B3n\\_de\\_dise%C3%B1o](http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o)

**Gracias**