Unidad 4: Diseño Orientado a Objetos

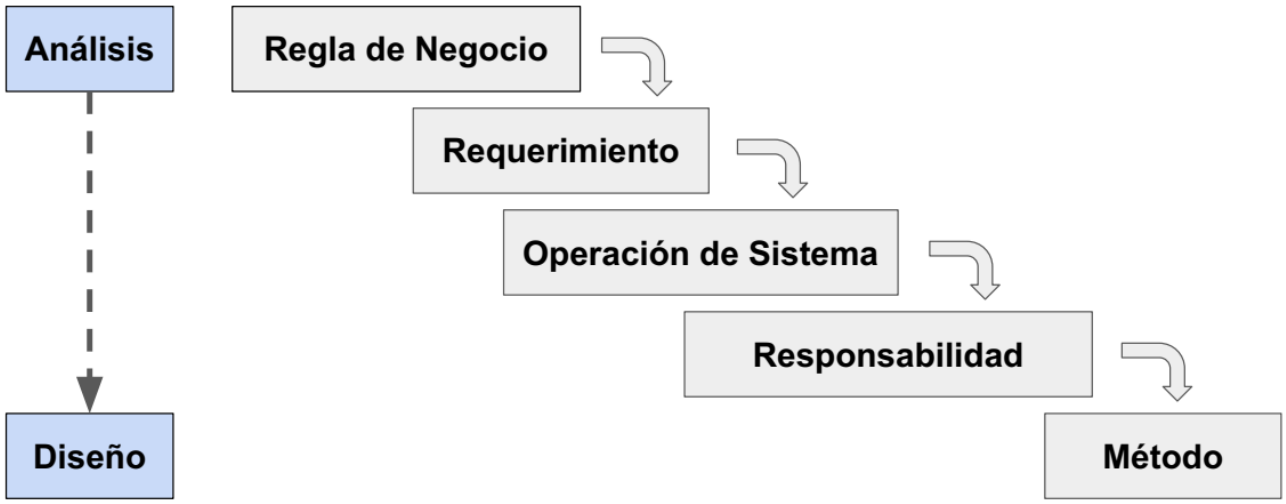
Indice

Diseño de aplicaciones OO

La OO hoy representa el mejor framework metodológico para la ingeniería de software gracias al pragmatismo del paradigma y la sistematización de procesos que permite.

- Homogeneidad a través del análisis, diseño e implementación.
- Énfasis en el estado, comportamiento e interacción de objetos.
- Maduración y patrones de prácticas.
- Variedad de técnicas, métodos, procesos, estándares, modelos, notaciones, herramientas, componentes, lenguajes, ambientes, ejemplos, comunidad, práctica y experiencia.
- Testing y métricas

- Artefactos mas relevantes:
  - Dinámicos: diagramas de interacción UML.
  - Estructural: diagramas de diseño de clases UML.
- Competencias mas relevantes:
  - Principios para la asignación de responsabilidades.
  - Principios y patrones de diseño.
- Si bien el UML se usa, no es lo mas relevante.



Larman

Principios de diseño

- Polimorfismos: The open close principle (OCP)
  - Un modulo debe estar abierto para su extensión, pero cerrado para su modificación.
- Contratos: The Liskov Substitution principle (LSP)
  - Las subclases deben ser sustituibles por sus clases de base
- Abstractas: The dependency inversión principle (DIP)
  - Dependa de las abstracciones. No dependa de las concreciones.
- Interfaces Simples: The interface segregation principle (ISP)
  - Es mejor tener muchas interfaces específicas del cliente que una sola de propósito general.

Pressman

Patrones GRASP

GRASP es un acrónimo de General Responsability Assignment Software Patterns. Los patrones GRASP describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones.

- Controlador.
- Experto en Información.
- Creador.
- Alta cohesión.
- Bajo acoplamiento.
- Polimorfismo.
- Fabricación pura.
- Indirección.
- Variaciones protegidas.

Controlador

- Problema:
  - Quien debe ser responsable de gestionar un evento de entrada del sistema (operación del sistema)?
- Solución:
  - Asignar la responsabilidad de manejar un mensaje de evento de entrada del sistema a una clase controladora que representa una parte del sistema.

Objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema, es decir, define un método para las operaciones del sistema.

- Representa el sistema global , dispositivo o un subsistema (controlador de fachada).
- Representa un escenario de casos de uso en el que tiene lugar el evento del sistema (controlador de caso de uso o sesión).

Experto en información.

- Problema:
  - Cual es el principio general para asignar responsabilidades a los objetos?
- Solución:
  - Asignarlas a la clase que tiene toda la información necesaria para realizar una responsabilidad, el experto de información.

Creador

- Problema:
    - Quien debería ser el responsable de la creación de una nueva instancia de alguna clase?
- Solución:
    - B es un creador de A, si se le asigna a la clase B la responsabilidad de crear una instancia de la clase A, y se cumple uno o mas de estos casos.

✓ B agrega objetos de A.

✓ B contiene objetos de A.

✓ B registra/persiste instancias de objetos de A.

✓ B utiliza mas estrechamente objetos de A.

✓ B tiene los datos de inicialización necesarios al momento de crear A.

Fabricación pura

- Problema:
    - ¿Quien es el responsable cuando está desesperado, y no quiere violar los principios de alta cohesión y bajo acoplamiento?
- Solución:
    - Asigne un conjunto altamente cohesivo de responsabilidades a una clase de “comportamiento” artificial o de conveniencia que no representa un concepto del dominio del problema (algo inventado), para dar soporte a la alta cohesión, bajo acoplamiento y la reutilización.

Larman

Lee el modelo del dominio donde aparecen los principales conceptos de la realidad del problema, y esos conceptos son las clases a utilizar.

Se implementa por que los patrones tradicionales no calzan correctamente en el problema a solucionar.

Es la razón del por que inventamos clases en nuestros diseños.

Son clases agregadas para dar soporte informático.

Indirección

- Problema:
  - ¿Como asignar responsabilidades para evitar el acoplamiento directo?
- Solución:
  - Asigne la responsabilidad a un objeto intermedio para medir entre otros componentes o servicios, de manera que no se acoplan directamente.

Larman

Si tengo un acoplamiento entre dos clases, se debe colocar un elemento intermedio general donde deslinde responsabilidades.

Las interfaces son una de las herramienta para implementar indirección. Puede también ser una clase.

Soriano

Variaciones protegidas

- Problema:
  - ¿Como asignar responsabilidades a los objetos, subsistemas, y sistemas de manera que las variaciones o inestabilidad en estos elementos no influya de manera no deseable en otros elementos?
- Solución:
  - Identifique los puntos de variaciones predecibles o inestabilidad; asigne las responsabilidades para crear una “interfaz” estable alrededor de ellos.

Larman

Apunta a la mantenibilidad y la flexibilidad. Se basa principio de sustitución de Liskov.

Se orienta a que ciertas funcionalidades de un sistema (que debe variar), se van a mantener constantes. REVISAR

A la hora de diseñar y asignar responsabilidades, hay que prestar atención a si cierta funcionalidad no se va a ver afectada o modificada.

Soriano

Alta cohesión

- Problema:
  - ¿Como mantener manejable la complejidad?
- Solución:
  - Asigne responsabilidades de manera que la cohesión permanezca alta.

Bajo acoplamiento

- Problema:
  - ¿Como dar soporte a las bajas dependencias y al incremento de la reutilización?
- Solución:
  - Asigne responsabilidades de manera que el acoplamiento (innecesario) se mantenga bajo.

Polimorfismo

- Problema:
  - ¿Quien es el responsable cuando el comportamiento varia en función del tipo?
- Solución:
  - Cuando las alternativas o comportamientos relacionados varían según el tipo (clase), asigne la responsabilidad del comportamiento (utilizando operaciones polimórficas) a los tipos para los que varia el comportamiento.

Patrones de diseño (GoF)

Un patrón de diseño es una descripción de clases y objetos comunicándose entre si, adaptada para resolver un problema de diseño general en un contexto particular.

Como usar un patrón de diseño?

1. Conocer en gran parte la solución genérica que brinda cada patrón y el tipo de problemática que resuelve.
2. Evaluar la posibilidad de aplicar el o los patrones en el problema correspondiente.
3. Definir la implementación de los mismos en nuestro modelo.
4. Implementar los métodos correspondientes.

By Purpose				
		Creational	**Structural**	**Behavioral**
By Scope	**Class**	• Factory Method	• Adapter (class)	• Interpreter • Template Method
	Object	• Abstract Factory • Builder • Prototype • Singleton	• Adapter (object) • Bridge • Composite • Decorator • Façade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

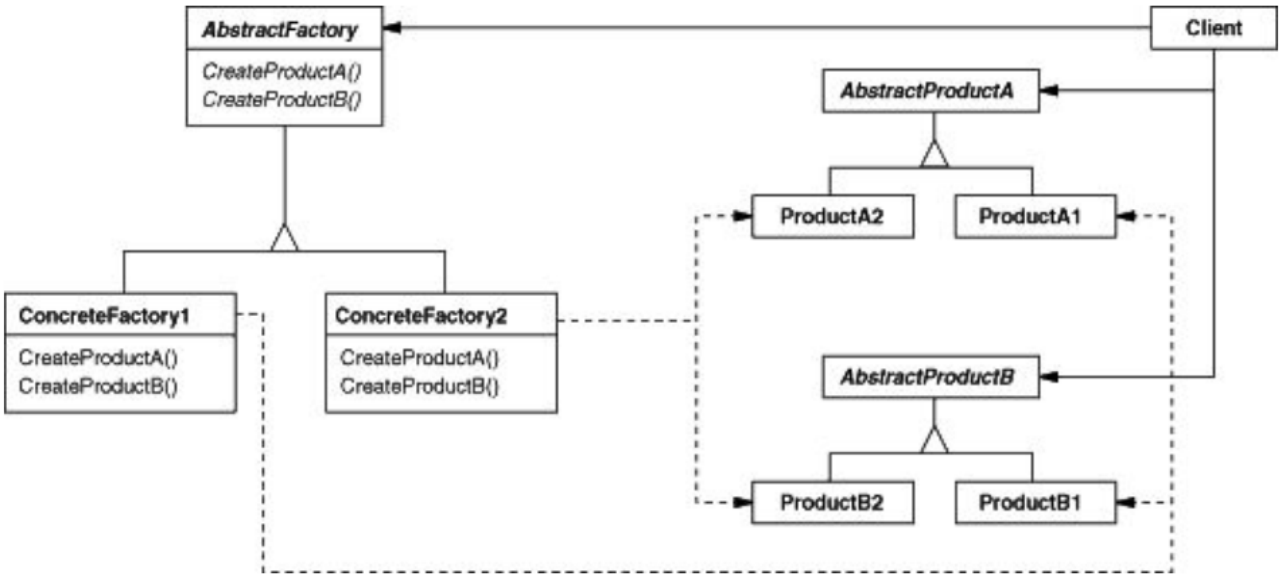
Abstract Factory (GoF)

Propósito:

Proporciona un interfaz para crear familias de objetos relacionados a que dependen entre si, sin especificar sus clases concretas.

Motivación:

- Un sistema podría ser independiente de como se crean, componen y representan sus productos.
- Un sistema podría ser configurado para usar una familia de productos entre varias.
- Una familia de objetos es cohesiva y se usan todos en conjunto.
- Se puede ofrecer una biblioteca de clase revelando solo sus interfaces.



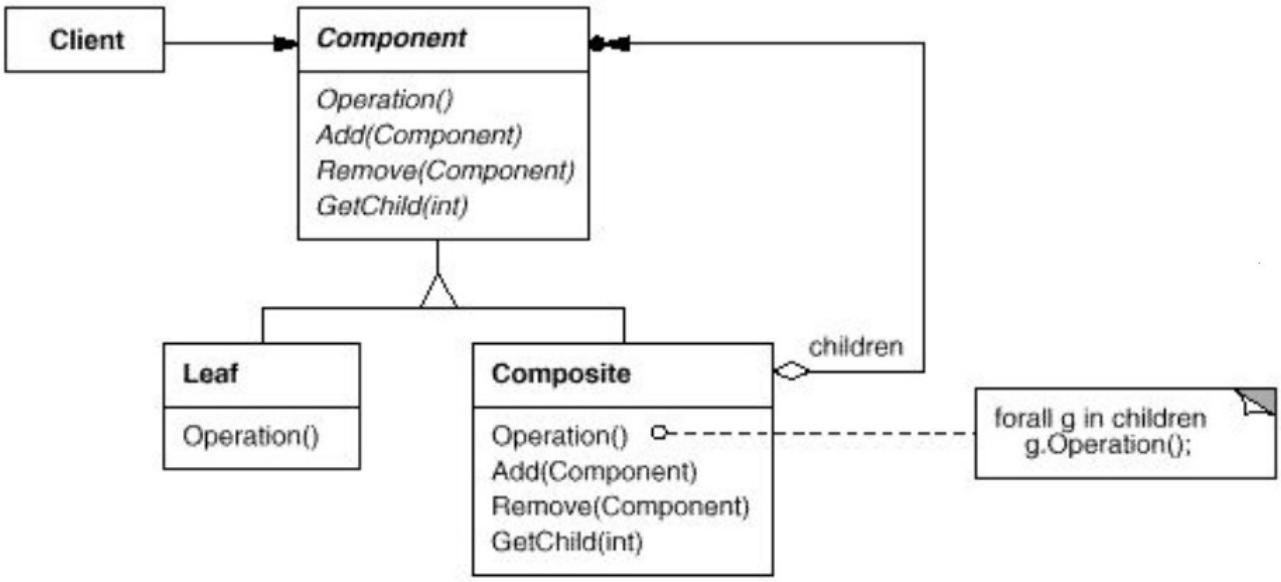
Composite (GoF)

Propósito:

Componer objetos de complejidad mayor mediante otros mas sencillos de forma recursiva.

Motivación:

- Cuando se necesite manejar de igual forma objetos sencillos o agrupaciones de estos.
- Se necesite representar jerarquías de objetos (árbol).



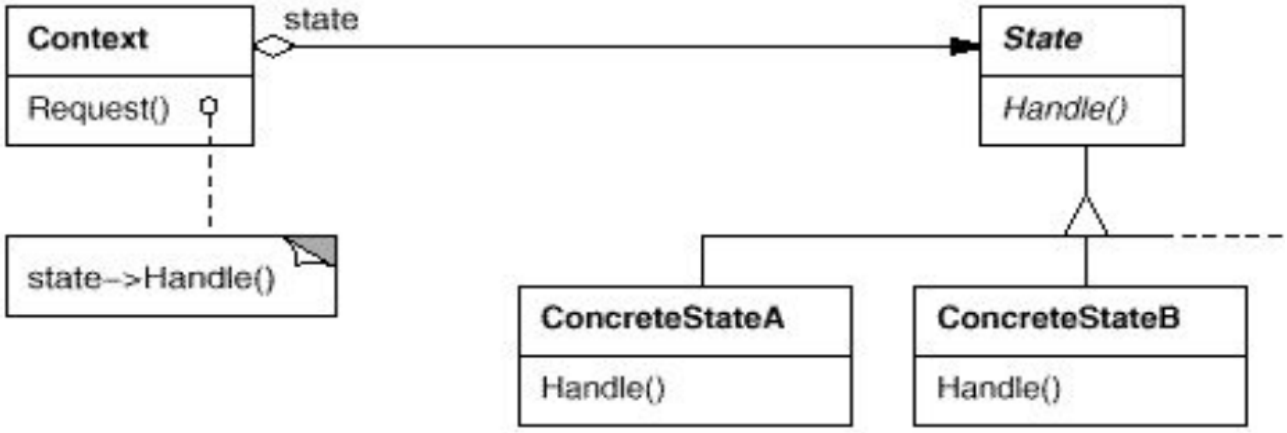
State (GoF)

Propósito:

Modelo para que un objeto cambie su comportamiento dependiendo de un estado.

Motivación:

- Cuando se necesita implementar comportamiento basado en estados o una maquina de estados (state diagram, UML).**<**



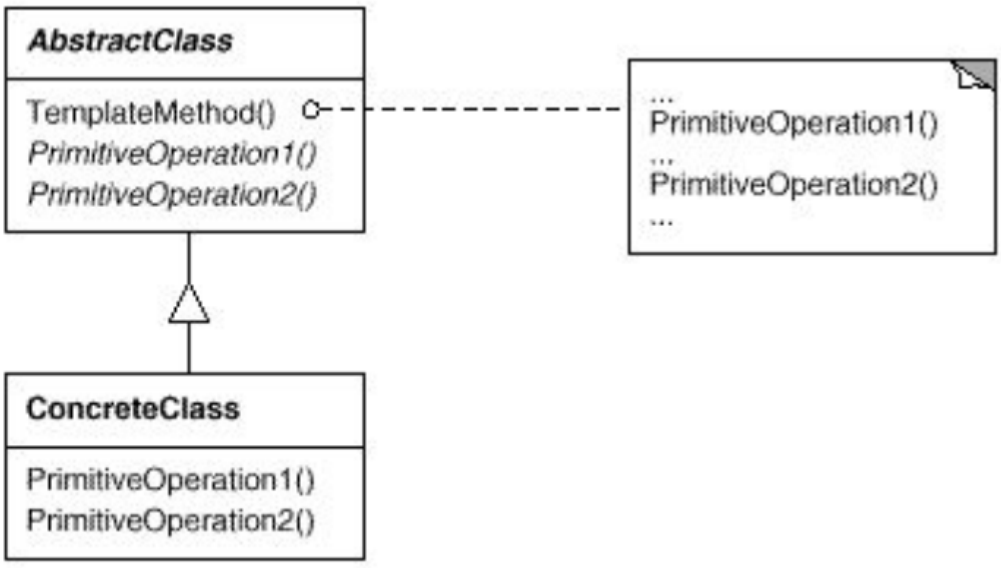
Template Method (GoF)

Propósito:

Definir el esqueleto de un algoritmo de un método, delegando tareas a las subclases para que redefinan o completen la funcionalidad pendiente.

Motivación:

Se puede implementar partes invariables de un algoritmo, dejando a las subclases que definan el comportamiento que puede variar.



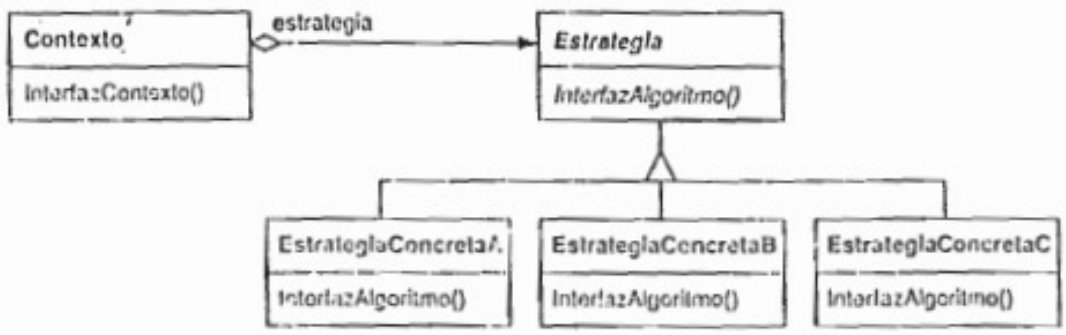
Strategy o Policy (GoF)

Propósito:

Define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan.

Motivación:

A veces pueden existir distintos algoritmos para resolver un mismo problema y dependiendo del objetivo deseado, se pueden priorizar uno u otro algoritmo. Por este motivo surge el strategy. Un claro ejemplo seria la utilización de un algoritmo de ordenamiento, donde dependiendo del caso, es mas útil un algoritmo que otro.



NOTA: Si bien puede resultar algo confuso la diferencia entre state y strategy, es preciso tener en cuenta de que state cambia **como** hacer una cosa de acuerdo al estado del objeto. Por otro lado el strategy cambia lo **que** se va a hacer de acuerdo a la estrategia tomada.

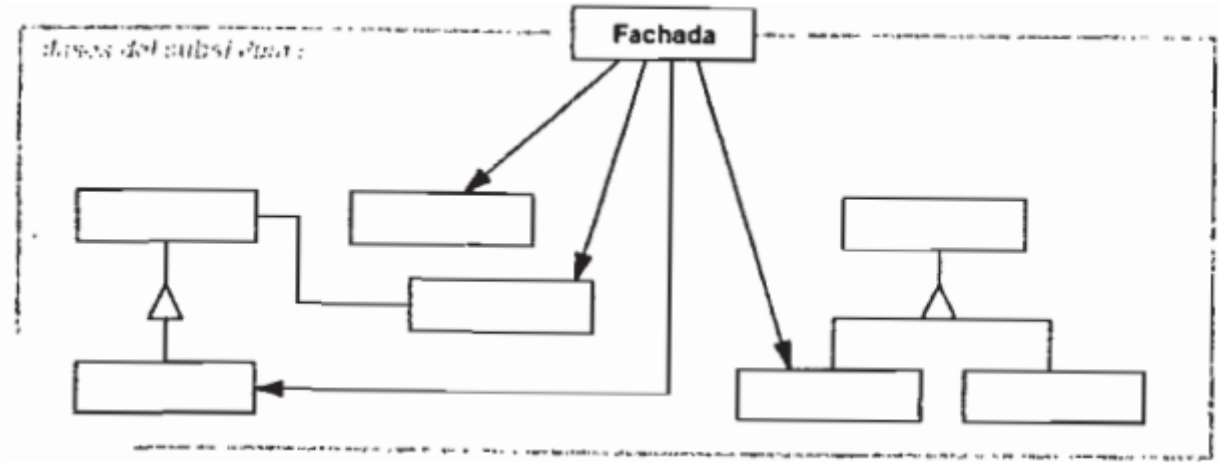
Facade o Fachada (GoF)

Propósito:

Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea mas fácil de usar.

Motivación:

Estructurar un sistema en subsistemas ayuda a reducir la complejidad. Un típico objetivo de diseño es minimizar la comunicación y dependencias entre subsistemas. Un modo de lograr esto es introduciendo un objeto fachada que proporcione una interfaz única y simplificada para los servicios mas generales del subsistema.



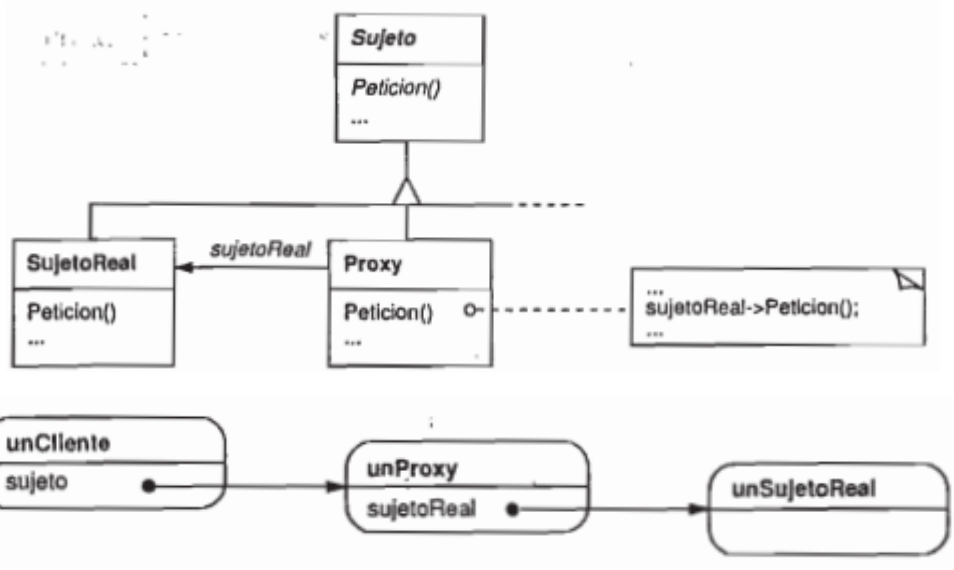
Proxy o Surrogate (GoF)

Propósito:

Proporciona un representante o sustituto de otro objeto para controlar el acceso a este.

Motivación:

Se utiliza para controlar ciertas acciones que solo serán disparadas en cierto momento y no en la creación del objeto.



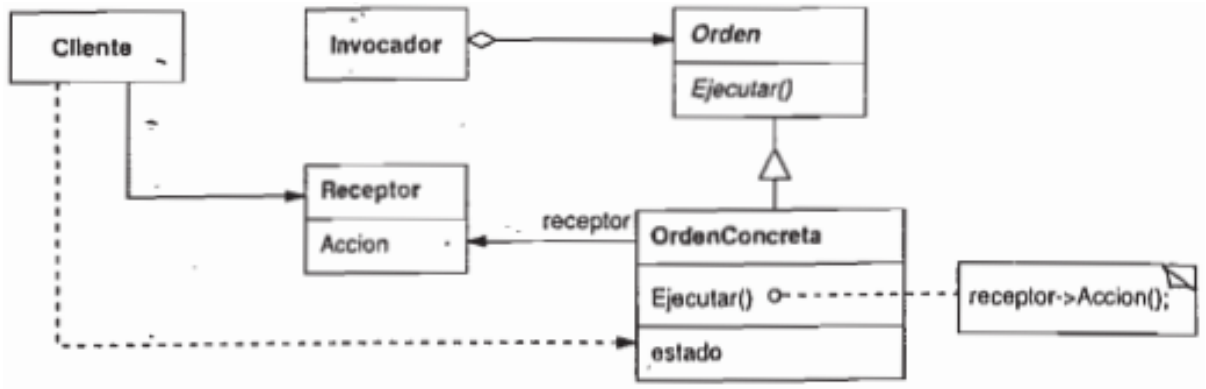
Command (GoF)

Propósito:

Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con diferentes peticiones, hacer cola o llevar un registro de las peticiones, y poder deshacer las operaciones.

Motivación:

A veces es necesario enviar peticiones a objetos sin saber nada acerca de la operación solicitada o de quien es el receptor de la petición.



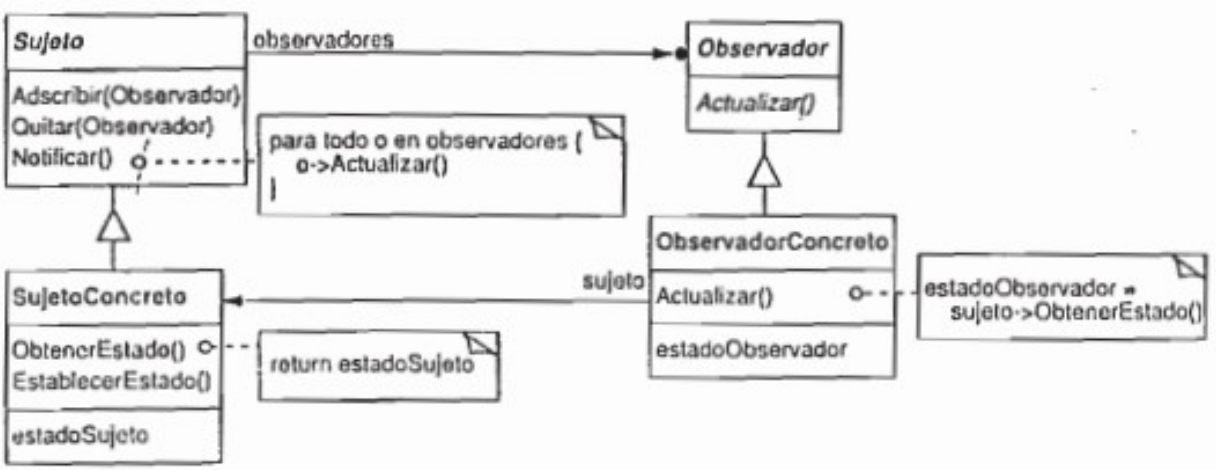
Observer (GoF)

Propósito:

Define una dependencia de uno a muchos entre objetos, de forma que cuando cambie de estado se notifique y se actualicen automáticamente todos los objetos que dependen de el.

Motivación:

Un efecto lateral habitual de dividir un sistema en una colección de clases cooperantes es la necesidad de mantener una consistencia entre objetos relacionados. No queremos alcanzar esa consistencia haciendo a las clases fuertemente acopladas, ya que eso reduciría su reutilización.



Framework

Un framework es una aplicación reusable, y “semi completa”, que puede ser especializada para integrarse a otra aplicación.

Es una colección de clases que en conjunto constituyen una solución genérica, a una familia de requerimientos de un dominio especifico.

- Objetivo: Reusabilidad.
  - Componer aplicaciones componiendo piezas de software reusables.
  - + Productividad, + Calidad, - Mantenimiento.
- Usos:
  - FMK por herencia (white box). Template Method (GoF).
  - FMK por composición (black box).
- Inversión de control.

Ventajas:

- Reutilización de código y diseño.
- Mejora la calidad.
- Puede simplificar la programación de un sistema.

Desventajas:

- Depende del lenguaje.
- Debug.
- Curva de aprendizaje e integración de múltiples frameworks.

Gamma

Refactoring

[…] para reestructurar software aplicando una serie de refactoreos sin cambiar su comportamiento observable.

Fowler

Visibilidad:

- Atributo:
  - Cuando dentro de un objeto existe una referencia a otro. En UML esto se representa como una referenciación.
- Local:
  - Cuando dentro de un objeto (un método) se genera la referencia al objeto. Es lo cal por que se define dentro de un método, y debería vivir dentro de ese método.
- Global:
  - Simplemente se accede a la variable, generalmente es un singleton (a nivel objeto) o se usan clases estáticas (a nivel clase).
- Paramétrica:
  - Recibir como parámetro en un método una referencia a un objeto.

Una referencia loca, global o paramétrica se representa en UML con una linea punteada (se muestra en los gráficos de las diapositivas de soriano).