2 ENERO 2013 LUISMI GRACIA

Un poco de Patrones de Diseño GoF (Gang of Four)

El objetivo principal de los patrones es facilitar la reutilización de diseños y arquitecturas software que han tenido éxito capturando la experiencia y haciéndola accesible a los no expertos.

Dentro de los patrones clásicos tenemos los **GoF (Gang of Four),**estudiados por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides en su mítico libro Design Patterns se contemplan 3 tipos de patrones:

· **Patrones de creación:** tratan de la inicialización y configuración de clases y objetos

· **Patrones estructurales**: Tratan de desacoplar interfaz e implementación de clases y objetos

· **Patrones de comportamiento** tratan de las interacciones dinámicas entre sociedades de clases y objetos

Y dentro de cada grupo tenemos:

Patrones de creación

· **Abstract Factory**. Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.

· **Builder**. Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.

· **Factory Method**. Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.

· **Prototype**. Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crear nuevos objetos copiando este prototipo.

· **Singleton**. Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.

Patrones estructurales

· **Adapter**. Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permiten que cooperen clases que de otra manera no podrían por tener interfaces incompatibles.

· **Bridge**. Desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.

· **Composite**. Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

· **Decorator**. Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.

· **Facade**. Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema se más fácil de usar.

· **Flyweight**. Usa el compartimiento para permitir un gran número de objetos de grano fino de forma eficiente.

· **Proxy**. Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.

Patrones de comportamiento

· **Chain of Responsibility**. Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada por algún objeto.

· **Command**. Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer la operaciones.

· **Interpreter**. Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.

· **Iterator**. Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.

· **Mediator**. Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.

· **Memento**. Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste puede volver a dicho estado más tarde.

· **Observer**. Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos.

· **State**. Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Parecerá que cambia la clase del objeto.

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

· **Template Method**. Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.

· **Visitor**. Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.

Si de este resumen os interesa este tema y os da pereza leer en inglés os recomiendo esta presentación:

Patrones de diseño orientado a objetos

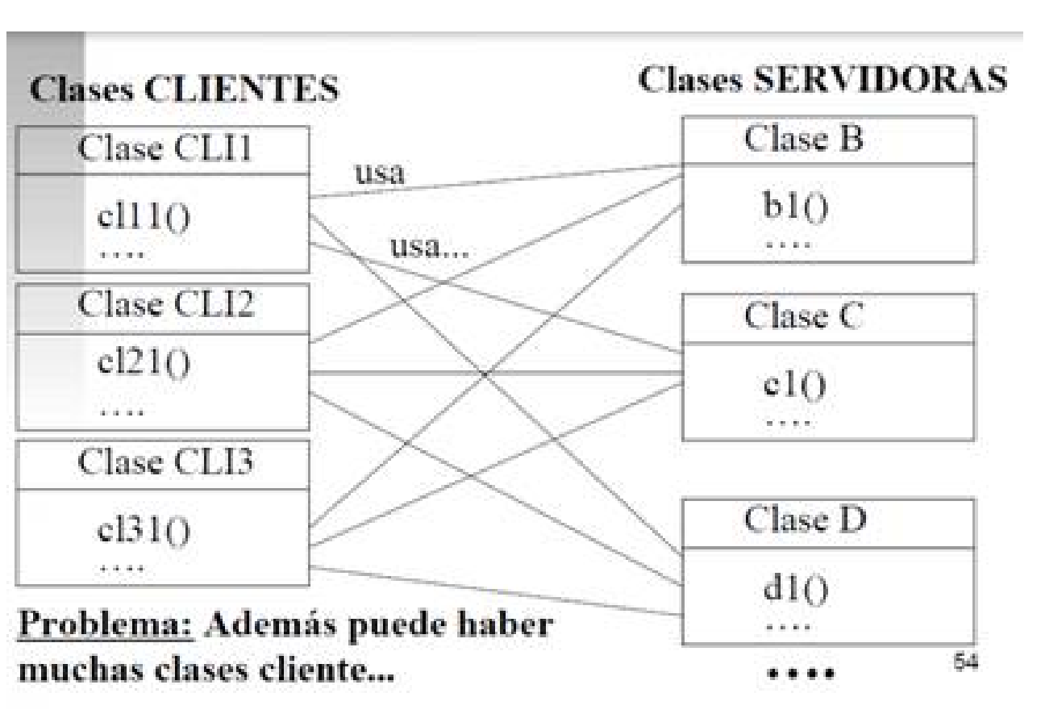
Programación Orientada a Objetos
Facultad de Informática

Juan Pavón Mestras
Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense Madrid

En este otro PDF se hace un estudio de estos mismos patrones mostrando ejemplos con código.

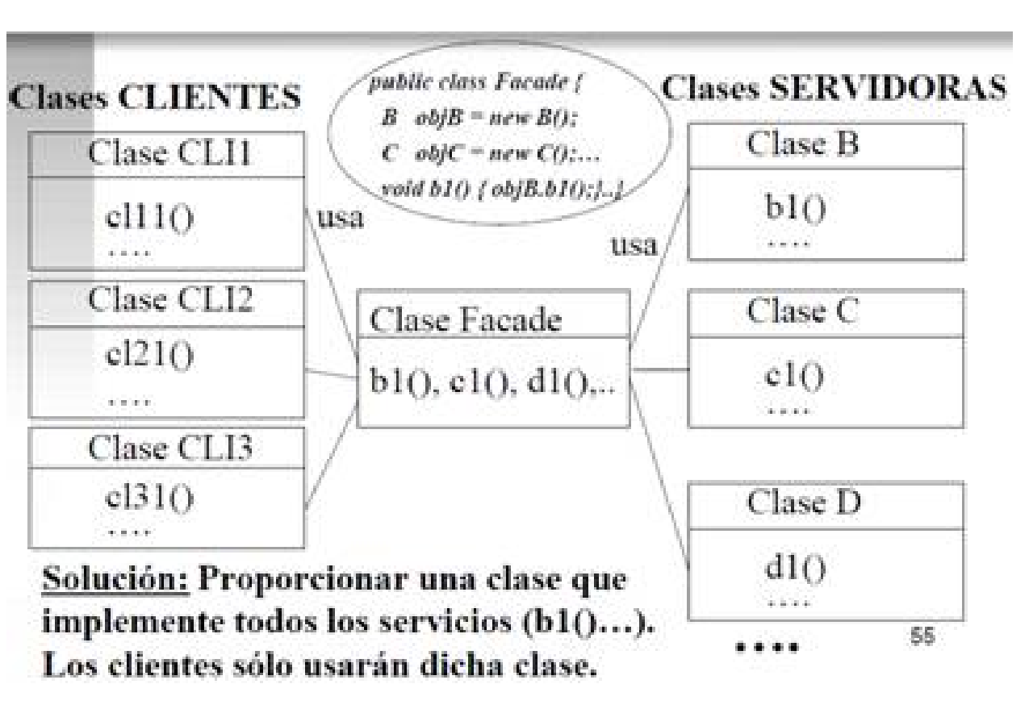
Por ejemplo para el patrón Facade:

El problema:



Problema: Además puede haber muchas clases cliente...

La solución:



```
public class Facade {
  B  objB = new B();
  C  objC = new C();...
  void b1() { objB.b1();}..}
```

Solución: Proporcionar una clase que implemente todos los servicios (b1()...). Los clientes sólo usarán dicha clase.

★★★★★ 5 Votes

Tu voto:

Comparte esto:

Cargando...

Relacionado

Jt7.0 – Framework para Android 12 noviembre 2010 En «Android»	Proxies en Java 14 May 2010 En «Java»	Patrones Estructurales: Escenarios de uso 6 May 2013 En «Arquitectura»

‹ Anterior
Y otra más de Visores JSON

Siguiente ›
Canonical presenta Ubuntu Phone

Deja un comentario

Buscar ...

Síguenos

Dirección de correo ele

Mail

Comentarios Rebloguear Suscribirse

suscriptores

Categorías

Elegir la categoría

Archivos

Elegir el mes

Sígueme en Twitter

Buscar ...

Síguenos

Dirección de correo ele

Mail

Únete a otros 1.049 suscriptores

Archivos

Elegir el mes

Categorías

Elegir la categoría

Sígueme en Twitter