

UNIVERSIDAD TECNOLÓGICA METROPOLITA
DEPARTAMENTO DE INFORMÁTICA
AYUDANTÍA DE INGENIERÍA DE SOFTWARE

Patrones de Diseño de Software

Alumnos:

David Martinez
Nicolas Paez
Ronald Verdugo

Ayudante:

Cristian Garrido

12 de Diciembre de 2013

¿Qué es un Patrón de Diseño?

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Debemos tener presente los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios).

Existen varios patrones de diseño popularmente conocidos, los cuales se clasifican como se muestra a continuación:

- **Patrones Creacionales:** Inicialización y configuración de objetos.
- **Patrones Estructurales:** Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes.
- **Patrones de Comportamiento:** Más que describir objetos o clases, describen la comunicación entre ellos.

Veamos un poco en qué consisten los distintos tipos de patrones, cuáles son sus fines y qué beneficios nos aportan.

1. Patrones Creacionales

Fábrica Abstracta (Abstract Factory)

El problema a solucionar por este patrón es el de crear diferentes familias de objetos, como por ejemplo la creación de interfaces gráficas de distintos tipos (ventana, menú, botón, etc.).

Método de Fabricación(Factory Method)

Parte del principio de que las subclases determinan la clase a implementar.

Prototipado (Prototype)

Se basa en la clonación de ejemplares copiándolos de un prototipo.

Singleton

Restringe la instanciación de una clase o valor de un tipo a un solo objeto.

MVC (Model View Controler)

Este patrón plantea la separación del problema en tres capas: la capa model, que representa la realidad; la capa controler , que conoce los métodos y atributos del modelo, recibe y realiza lo que el usuario quiere hacer; y la capa vista, que muestra un aspecto del modelo y es utilizada por la capa anterior para interaccionar con el usuario.

2. Patrones Estructurales

- **Adaptador (Adapter):** Convierte una interfaz en otra.
- **Puente (Bridge):** Desacopla una abstracción de su implementación permitiendo modificarlas independientemente.
- **Objeto Compuesto (Composite):** Utilizado para construir objetos complejos a partir de otros más simples, utilizando para ello la composición recursiva y una estructura de árbol.

- Envoltorio (Decorator): Permite añadir dinámicamente funcionalidad a una clase existente, evitando heredar sucesivas clases para incorporar la nueva funcionalidad.
- Fachada (Facade): Permite simplificar la interfaz para un subsistema.
- Peso Ligero (Flyweight): Elimina la redundancia o la reduce cuando tenemos gran cantidad de objetos con información idéntica.
- **Proxy (Este Patron se detallará):** Un objeto se aproxima a otro.

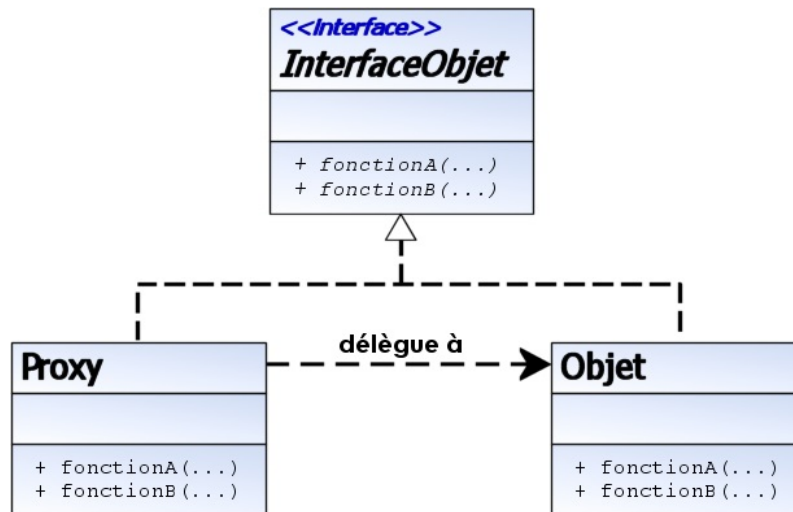
El patrón proxy se usa cuando se necesita una referencia a un objeto más flexible o sofisticada que un puntero. Dependiendo de la función que se desea realizar con dicha referencia podemos distinguir diferentes tipos de proxies:

proxy remoto: representante local de un objeto remoto.

proxy virtual: crea objetos costosos bajo demanda (como la clase ImagenProxy en el ejemplo de motivación).

proxy de protección: controla el acceso al objeto original.

proxy de referencia inteligente: sustituto de un puntero que lleva a cabo operaciones adicionales cuando se accede a un objeto (ej. contar número de referencias al objeto real, cargar un objeto persistente bajo demanda en memoria, control de concurrencia de acceso tal como bloquear el objeto para impedir acceso concurrente,).



3. Patrones de Comportamiento

- Cadena de responsabilidad (Chain of responsibility): La base es permitir que más de un objeto tenga la posibilidad de atender una petición.
- Orden (Command): Encapsula una petición como un objeto dando la posibilidad de deshacer la petición.
- Intérprete (Interpreter): Intérprete de lenguaje para una gramática simple y sencilla.
- Iterador (Iterator): Define una interfaz que declara los métodos necesarios para acceder secuencialmente a una colección de objetos sin exponer su estructura interna.

- Mediador (Mediator): Coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.
- Recuerdo (Memento): Almacena el estado de un objeto y lo restaura posteriormente.
- Observador (Observer): Notificaciones de cambios de estado de un objeto.
- Estado (Server): Se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo.
- Estrategia (Strategy): Utilizado para manejar la selección de un algoritmo.
- Método plantilla (Template Method): Algoritmo con varios pasos suministrados por una clase derivada.
- Visitante (Visitor): Operaciones aplicadas a elementos de una estructura de objetos heterogénea.

Conclusiones

- Los patrones de diseño describen la solución a problemas que se repiten una y otra vez en nuestros sistemas, de forma que se puede usar esa solución siempre que haga falta.
- Capturan el conocimiento que tienen los expertos a la hora de diseñar.
- Ayudan a generar software maleable (software que soporta y facilita el cambio, la reutilización y la mejora).
- Son guías de diseño, no reglas rigurosas.

https://github.com/ronaldvl/AyudantiaSw_Tareas