XPERTYS
CONSULTORÍA & FÁBRICA DE SOFTWARE

Centro de Conocimiento

Patrones de Diseño

Descripciones de los patrones de diseño.

Esta guía proporciona un resumen de los patrones de diseño, basada en el libro Design Patterns:

Elements of Reusable Object-Oriented Software.

También incluye diagramas de clases, explicación, información de uso y ejemplos aplicados al mundo real.

Patrones de Diseño

Acerca de los patrones de Diseño

Esta guía proporciona un resumen de los patrones de diseño, basada en el libro *Design Patterns: Elements of Reusable Object-Oriented Software*. También incluye diagramas de clases, explicación, información de uso y ejemplos aplicados al mundo real.

Clasificación

Patrones Creacionales : Se utilizan para construir objetos de tal manera que pueden ser desacoplados de su sistema de implementación.

Patrones Estructurales: Se utilizan para formar grandes estructuras de objetos entre muchos objetos diferentes.

Patrones de Comportamiento: Se utiliza para manejar algoritmos, relaciones y responsabilidades entre objetos.

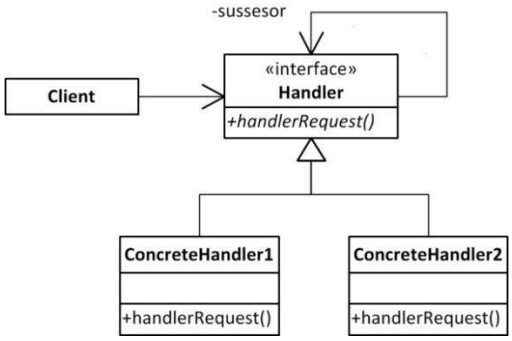
Alcance de objeto: Trata de la relaciones de un objeto que pueden ser cambiadas en tiempo de ejecución.

Alcance de Clase: Trata de las relaciones de una clase que se pueden cambiar en tiempo de compilación.

Patrones Creacionales	Patrones Estructurales	Patrones de Comportamiento
• Abstract Factory	• Adapter	• Chain of Responsibility
• Builder	• Bridge	• Command
• Factory Method	• Composite	• Interpreter
• Prototype	• Decorator	• Iterator
• Singleton	• Facade	• Mediator
	• Flyweigth	• Memento
	• Proxy	• Observer
		• State
		• Strategy
		• Template Method
		• Visitor

Patrones Creacionales

Chain of Responsability



Propósito

Proporciona a más de un objeto la oportunidad de manejar una petición por medio de una vinculación que recibe objetos y los relaciona.

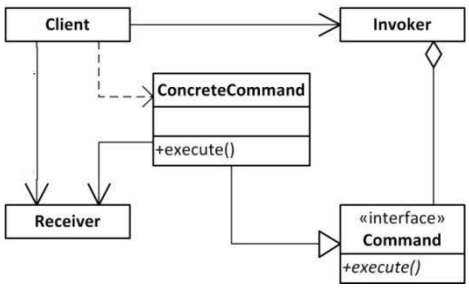
Se utiliza cuando

- Varios objetos pueden manejar una petición y el controlador no tiene que ser un objeto específico.
- Un conjunto de objetos debe ser capaz de manejar una solicitud, con el controlador determinándolo en tiempo de ejecución.
- Una solicitud no siendo maneja da es un resultado potencialmente aceptado.

Ejemplo

El manejo de excepciones en algunos idiomas implementa este patrón. Cuando se produce una excepción en tiempo de ejecución comprueba si el método tiene un mecanismo para controlar la excepción o si se debe pasar por una pila de llamadas. Cuando se le pasa por la pila de llamadas el proceso se repite hasta que el código que maneja la excepción es encontrado o asta que no haya mas objetos que puedan manejar la excepción.

Commmad



Propósito

Encapsula una solicitud que le permite ser tratada como un objeto. Esto permite que la solicitud sea manejada de manera tradicional en objeto, con relaciones basadas tales como colas y devoluciones de llamadas.

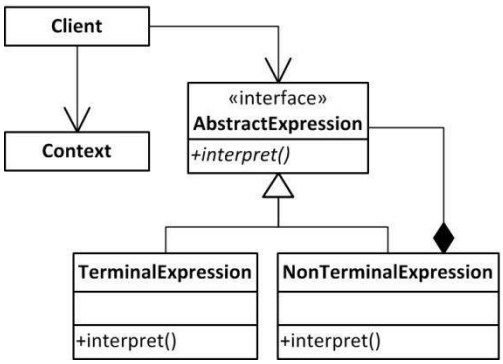
Se utiliza cuando

- Necesitas funcionalidad de devolución de llamada.
- Las solicitudes deben de ser manejadas en orden o tiempo variante.
- Se necesita un historial de peticiones.
- El invocador debe desacoplarse del objeto manejando la invocación.

Ejemplo

Las colas de trabajo son ampliamente utilizadas para facilitar los procesos asíncronos de algoritmos. Al utilizar el patrón la funcionalidad a ejecutar se puede dar a una cola de trabajo para procesamiento sin ninguna necesidad de conocimiento por parte de la cola del invocador que actualmente se esta invocando. El comando que se pone en cola implementa su algoritmo en particular encapsulándolo de la interfaz que la cola espera.

Interpreter



Propósito

Define una representación de una gramática, si como un mecanismo para comprender y actuar sobre la gramática.

Se utiliza cuando

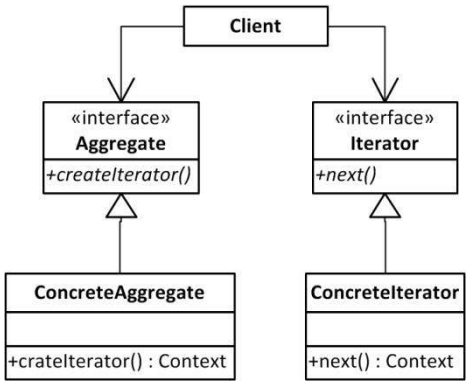
- Existe una gramática a interpretar que puede representar una gran sintaxis en forma de árbol.
- La gramática es sencilla.
- La eficiencia no es importante.
- Se desea desacoplamiento en la gramática de expresiones subyacentes.

Ejemplo

Aventuras basadas en texto, muy populares en la

década de 1980, un buen ejemplo de ello. Muchos tenían órdenes sencillas, como "reducir" que permitió recorrido del juego. Estos comandos pueden ser anidados tal que altera su significado. Por ejemplo, "ir" daría lugar a un resultado diferente a "subir". Mediante la creación de una jerarquía de órdenes basándose en el comando y el calificador (expresiones no-terminal y terminal) la aplicación puede asignar fácilmente muchas variaciones de comandos a un árbol sobre las acciones.

ITERATOR



Propósito

Permite el acceso a los elementos de un objeto agregado sin permitir el acceso a la representación subyacente.

Se utiliza cuando

- Se necesita acceso a los elementos que no tienen acceso a la totalidad de la representación.
- Se necesitan múltiples o concurrente recorridos de los elementos.
- Se necesita una interfaz uniforme para el recorrido.
- Existen diferencias sutiles entre los detalles de la implementación de varios iteradores.

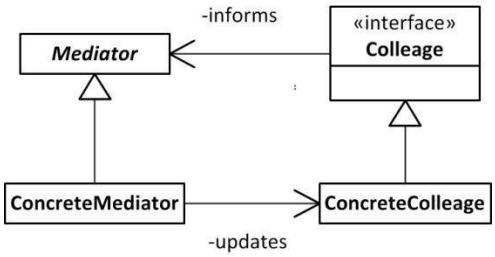
Ejemplo

La implementación de Java del patrón permite a los usuarios recorrer varios tipos de conjuntos de datos sin tener que preocuparse acerca de la implementación subyacente de la colección.

Ya que los clientes simplemente interactuar con la interfaz iterator, las colecciones se quedan para definir el repetidor adecuado para ellos.

Algunos permiten el acceso completo a los datos subyacentes establecidos mientras que otras pueden restringir ciertas funcionalidades, como la eliminación de elementos.

MEDIATOR



Propósito

Permite acoplamiento débil mediante la encapsulación de la forma en conjuntos dispares de los objetos interactúan y se comunican entre sí. Permite a las acciones de cada objeto creado para variar de forma independiente el uno del otro.

Se utiliza cuando

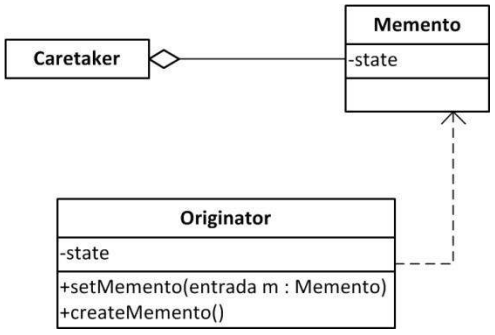
- La comunicación entre los conjuntos de objetos está bien definido y complejo
- Existen demasiadas relaciones y se necesita punto de control o de comunicación común.

Ejemplo

Software de lista de correo hace un seguimiento de quién está suscrito a la lista de correo y proporciona un único punto de acceso a través del cual cualquier persona puede comunicarse con la lista completa.

Sin una implementación mediador una persona que desea enviar un mensaje al grupo tendría que monitorear continuamente las que fue firmada y que no fue. Mediante la implementación del patrón de mediador el sistema es capaz de recibir mensajes desde cualquier punto a continuación, determinar qué receptores que transmita el mensaje en el que, sin que el remitente del mensaje que tiene que ser que se trate con la lista de destinatarios real.

MEMENTO

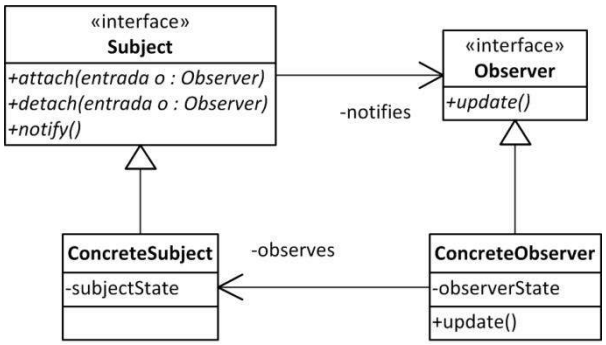


Propósito
Permite para capturar y externalización de estado interna de un objeto de forma que puede ser restaurado más adelante, todo sin violar la encapsulación.

Se utiliza cuando
- El estado interno de un objeto debe ser guardado y restaurado en un momento posterior.
- Interior del Estado no puede ser expuesto por interfaces sin exponer la aplicación.
- Límites de encapsulación se deben preservar.

Ejemplo
Deshacer funcionalidad muy bien puede ser implementado utilizando el patrón de recuerdo. Al serializar y deserializar el estado de un objeto antes de que ocurra el cambio que podemos conservar una instantánea de lo que más tarde puede ser restaurado si el usuario decide cancelar la operación.

OBSERVER

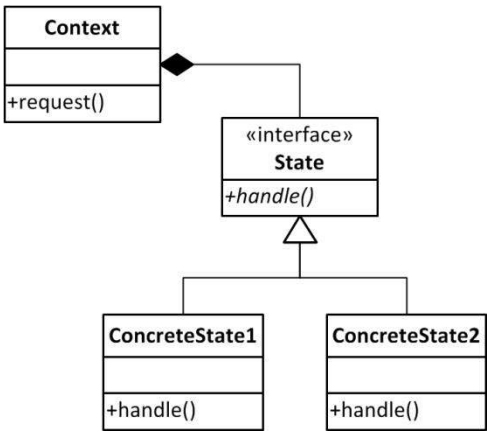


Propósito
Permite a uno o más objetos serán notificados de los cambios de estado de otros objetos en el sistema.

Se utiliza cuando
- Los cambios de estado de uno o varios objetos deben dar lugar a comportamientos de otros objetos
- Se requieren capacidades de Radiodifusión.
- El conocimiento existe de que los objetos estarán ciegos a expensas de la notificación.

Ejemplo
Este modelo se puede encontrar en casi todos los ambientes GUI. Cuando los botones, texto, y otros campos se colocan en aplicaciones de la aplicación normalmente se registra como listener para esos controles. Cuando un usuario activa un evento, como hacer clic en un botón, el control se repite a través de sus observadores registrados y envía una notificación a cada uno.

STATE

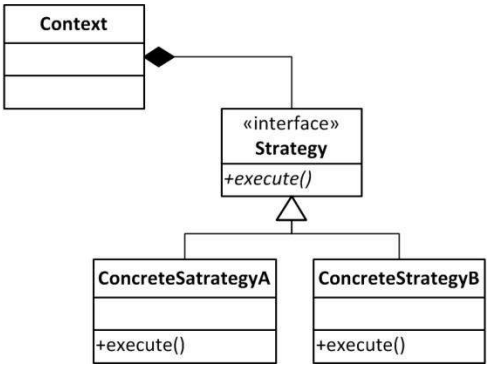


Propósito
Lazos objeto circunstancias a su comportamiento, lo que permite el objeto a comportarse de maneras diferentes sobre la base de su estado interno.

Se utiliza cuando
- El comportamiento de un objeto debe estar influenciada por su estado.
- Condiciones complejas atan comportamiento del objeto a su estado.
- Las transiciones entre estados deben ser explícitos.

Ejemplo
Un objeto de correo electrónico puede tener varios estados, todos los cuales va a cambiar la forma del objeto se encarga de diferentes funciones. Si el estado es "no enviado", entonces la llamada a send() va a enviar el mensaje durante una llamada a recallMessage() o bien se producirá un error o no hacer nada. Sin embargo, si el estado es "enviado", entonces la llamada a send() sería o generar un error o no hacer nada, mientras que la llamada a recallMessage() intentaría enviar una notificación de retirada a los destinatarios. Para evitar sentencias condicionales en la mayoría de o todos los métodos no habría múltiples objetos de estado que se encargan de la aplicación con respecto a su estado en particular. Las llamadas dentro del objeto Email entonces serían delegadas hasta el objeto de estado correspondiente para su resolución.

STRATEGY



Propósito

Define un conjunto de algoritmos encapsulados que se pueden intercambiar para llevar a cabo una determinada conducta.

Se usa cuando

- La única diferencia entre muchas clases relacionadas es su comportamiento.
- Se requieren varias versiones o variaciones de un algoritmo.
- Algoritmos de acceder o utilizar los datos que el código de llamada no debe ser expuesto.
- El comportamiento de una clase debe definirse en tiempo de ejecución.
- Sentencias condicionales son complejos y difíciles de mantener.

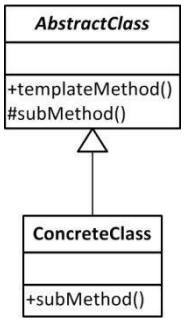
Ejemplo

Al importar datos a un nuevo sistema de diferentes algoritmos de validación puede ejecutarse basándose en el conjunto de datos.

Mediante la configuración de la importación de utilizar estrategias de la lógica condicional para determinar qué conjunto de validación para correr se puede quitar y la importación se puede desacoplar del código de validación actual.

Esto nos permitirá llamar dinámicamente una o más estrategias durante la importación.

TEMPLATE METHOD



Propósito

Identifica el marco de un algoritmo, que permite definir las clases que implementan el comportamiento real.

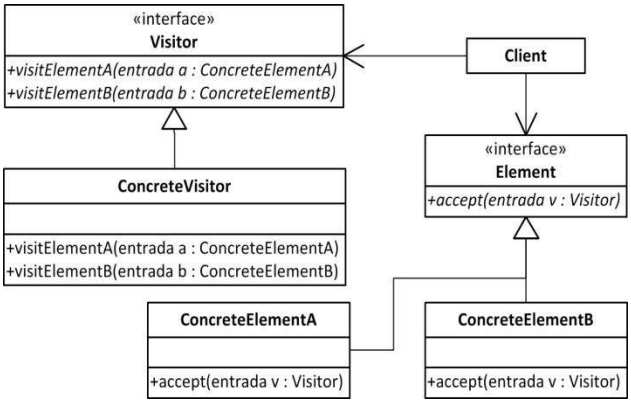
Se utiliza cuando

- Se necesita una sola aplicación de un algoritmo de resumen.
- Comportamiento común entre las subclases se debe traducir a una clase común.
- Clases de padres deben ser capaces de invocar de manera uniforme el comportamiento de sus subclases.
- La mayoría o todas las subclases necesitan para implementar el comportamiento.

Ejemplo

Una clase padre, InstantMessage, probablemente tendrá todos los métodos necesarios para gestionar el envío de un mensaje. Sin embargo, la serialización real de los datos a enviar puede variar dependiendo de la implementación. Un mensaje de video y un mensaje de texto sin formato requerirán diferentes algoritmos para serializar los datos correctamente. Las subclases de InstantMessage pueden proporcionar su propia implementación del método de serialización, lo que la clase padre para trabajar con ellos sin entender sus detalles de implementación.

VISITOR



Propósito

Permite una o más de las operaciones que han de aplicarse a un conjunto de objetos en tiempo de ejecución, las operaciones de desacoplamiento de la estructura del objeto.

Se utiliza cuando

- Una estructura de objeto debe tener muchas

operaciones relacionadas realizadas en ella.

- La estructura del objeto no se puede cambiar, pero las operaciones se realiza en lo posible.
- Las operaciones se deben realizar en las clases concretas de una estructura de objeto.
- La exposición de estado interno o de las operaciones de la estructura del objeto es aceptable.
- Las operaciones deben ser capaces de operar en múltiples estructuras de objetos que implementan los mismos conjuntos de interfaces.

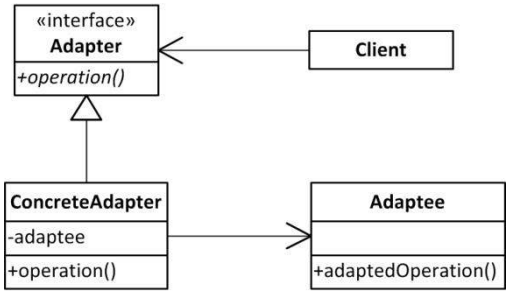
Ejemplo

Cálculo de impuestos en las diferentes regiones en un conjunto de facturas requeriría muchas variaciones diferentes de la lógica de cálculo.

La implementación de un visitante permite que la lógica que se desacopla de las facturas y partidas.

Esto permite que la jerarquía de los elementos a ser visitado por código de cálculo que luego pueden aplicar las tarifas adecuadas para la región. Cambio de regiones es tan simple como la sustitución de un visitante diferente.

ADAPTER



Propósito

Permite clases con interfaces diferentes para trabajar juntos por la creación de un objeto común mediante el cual pueden comunicarse e interactuar.

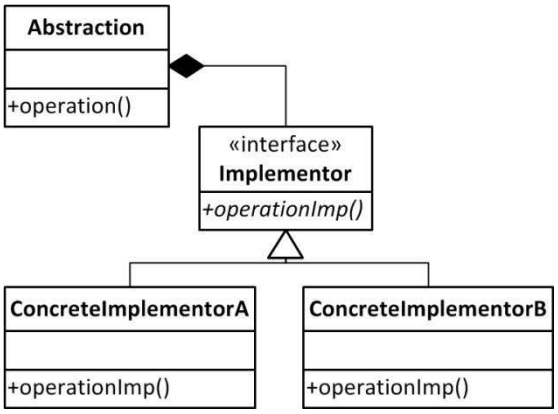
Se utiliza cuando

- Una clase para ser utilizado no cumple con los requisitos de interfaz.
- Condiciones complejas atan comportamiento del objeto a su estado.
- Las transiciones entre estados deben ser explícitos.

Ejemplo

Una aplicación de facturación tiene que interactuar con una aplicación de recursos humanos con el fin de intercambiar datos de los empleados, sin embargo cada uno tiene su propia interfaz y la implementación del objeto Empleado. Además, el SSN se almacena en diferentes formatos por cada sistema. Mediante la creación de un adaptador que podemos crear una interfaz común entre las dos aplicaciones que les permite comunicarse con sus objetos nativos y es capaz de transformar el formato de número de Seguro Social en el proceso.

BRIDGE



Propósito

Define una estructura de objeto abstracto independientemente de la estructura objeto de implementación con el fin de limitar el acoplamiento.

Se utiliza cuando

- Las abstracciones e implementaciones no deben ser obligados en tiempo de compilación.
- Las abstracciones e implementaciones deben ser independientemente extensible.
- Los cambios en la implementación de una abstracción no deben tener ningún impacto en los clientes.
- Los detalles de implementación deben ser ocultados al cliente.

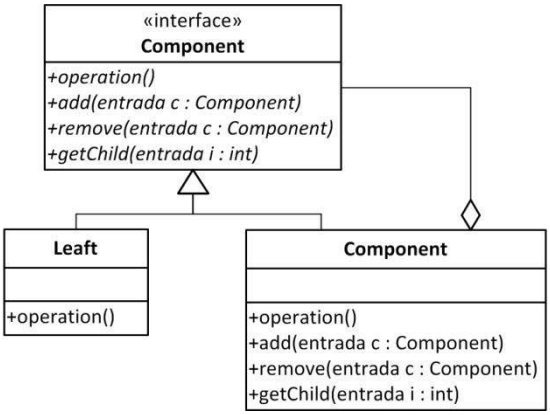
Ejemplo

La Máquina Virtual Java (JVM) tiene su propio conjunto nativo de funciones que resumen el uso de ventanas, el registro del sistema y la ejecución de código de bytes, pero la aplicación real de estas funciones se delega en el sistema operativo de la JVM se ejecuta.

Cuando una aplicación se encarga de la JVM para hacer una ventana delega la llamada prestación a la aplicación concreta de la JVM que sabe cómo

comunicarse con el sistema operativo con el fin de hacer que la ventana.

COMPOSITE



Propósito

Facilita la creación de jerarquías de objetos donde cada objeto puede ser tratado de forma independiente o como un conjunto de objetos anidados a través de la misma interfaz.

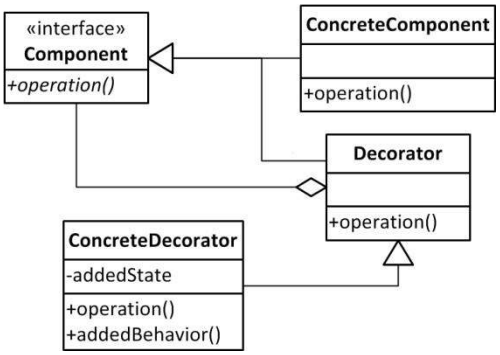
Se utiliza cuando

- Se necesitan representaciones jerárquicas de objetos.
- Objetos y composiciones de objetos deben ser tratados de manera uniforme.

Ejemplo

A veces, la información que se muestra en un carrito de compras es el producto de un solo punto, mientras que otras veces es una agregación de varios elementos. Mediante la implementación de los artículos como composites podemos tratar a los agregados y los puntos de la misma manera, lo que nos permite iterar simplemente sobre el árbol e invocamos la funcionalidad en cada artículo. Llamando al método getCost() en cualquier nodo dado que iba a conseguir el coste de ese artículo, más el costo de todos los elementos secundarios, permitiendo que los artículos a ser tratados de manera uniforme si eran artículos o grupos de artículos.

DECORATOR



Propósito

Permite la envoltura dinámica de objetos con el fin de modificar sus responsabilidades y comportamientos existentes.

Se utiliza cuando

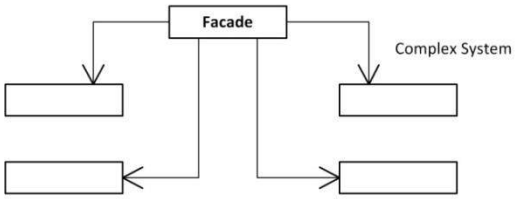
- Responsabilidades de objetos y comportamientos deben ser dinámicamente modificables.
- Implementaciones concretas deben desvincularse de las responsabilidades y los comportamientos.
- Subclases para lograr la modificación es imposible o poco práctico.
- La funcionalidad específica no debe residir altos en la jerarquía de objetos.
- Una gran cantidad de pequeños objetos que rodean una aplicación concreta es aceptable.

Ejemplo

Muchas empresas establecen sus sistemas de correo de tomar ventaja de los decoradores. Cuando los mensajes se envían a alguien en la empresa a una dirección externa al servidor de correo decora el mensaje original con derechos de autor y la información de carácter confidencial. Mientras el mensaje permanece interna de la información no está unida.

Esta decoración permite que el mensaje en sí se mantenga sin cambios hasta que se tome una decisión en tiempo de ejecución para envolver el mensaje con información adicional.

FACADE



Propósito

Proporciona una única interfaz a un conjunto de interfaces dentro de un sistema.

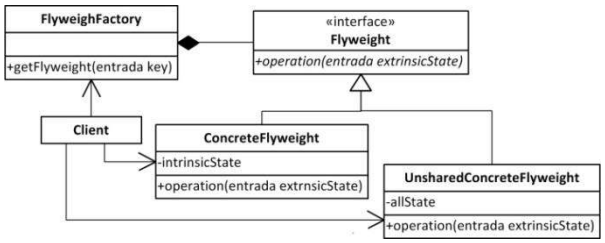
Se utiliza cuando

- Se necesita una interfaz sencilla para facilitar el acceso a un sistema complejo.
- Hay muchas dependencias entre las implementaciones de sistemas y clientes.
- Sistemas y subsistemas deben ser capas.

Ejemplo

Al exponer un conjunto de funcionalidades a través de un servicio web el código de cliente debe preocuparse sólo por la sencilla interfaz de la exposición a ellos y no a las complejas relaciones que pueden o no pueden existir detrás de la capa de servicios web. Una sola llamada de servicio web para actualizar un sistema con nuevos datos en realidad puede implicar la comunicación con una serie de bases de datos y sistemas, detalle se oculta debido a la implementación del patrón fachada.

FLYWEIGHT



Propósito

Facilita la reutilización de muchos objetos de grano fino, por lo que la utilización de un gran número de objetos de forma más eficiente.

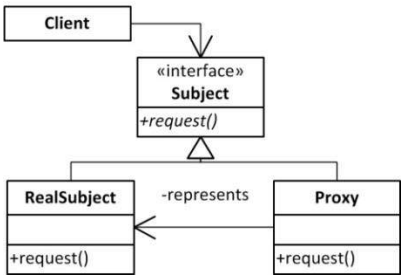
Se utiliza cuando

- Muchos, como los objetos se utilizan y el costo de almacenamiento es alto.
- La mayoría de los estados de cada objeto se puede convertir extrínseca.
- Unos objetos compartidos pueden remplazar muchos otros no compartidos.
- La identidad de cada objeto, no importa.

Ejemplo

Los sistemas que permiten a los usuarios definir sus propios flujos de aplicaciones y diseños a menudo tienen la necesidad de realizar un seguimiento de un gran número de campos, páginas y otros elementos que son casi idénticos entre sí. Al hacer estos artículos en contrapesos todas las instancias de cada objeto pueden compartir el estado intrínseco, manteniendo el estado extrínseco separado. El estado intrínseco sería almacenar las propiedades compartidas, como el aspecto de un cuadro de texto, la cantidad de datos que puede contener, y qué eventos se expone. El estado extrínseco sería almacenar las propiedades no compartidas, como la que pertenece el artículo, cómo reaccionar a un clic del usuario, y la forma de manejar los eventos.

PROXY



Propósito

Permite el control de acceso a nivel de objeto, actuando como un paso a través de la entidad o de un objeto marcador de posición.

Se utiliza cuando

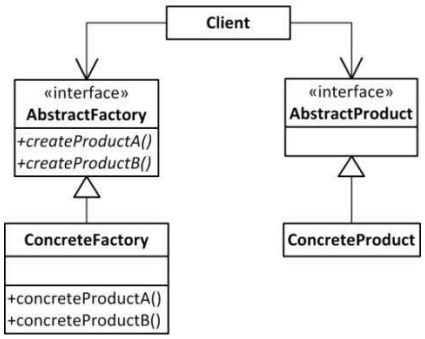
- El objeto que está siendo representada es externa al sistema.
- Los objetos se deben crear en la demanda.
- Se requiere un control de acceso para el objeto original.
- Se requiere funcionalidad Añadido cuando se accede a un objeto.

Ejemplo

Bitácora de aplicaciones a menudo proporcionan una forma para que los usuarios puedan conciliar sus estados de cuenta con sus datos del libro mayor de la demanda, la automatización de gran parte del

proceso. El funcionamiento real de la comunicación con un tercero es una operación relativamente cara que debe ser limitada. Mediante el uso de un proxy para representar el objeto de comunicación se puede limitar el número de veces o los intervalos se invocan la comunicación. Además, podemos envolver el complejo de instancias del objeto de comunicación dentro de la clase de proxy, la disociación código de llamada de los detalles de implementación.

Abstract Factory



Propósito

Proporcionar una interfaz que llama a la creación de delegados a una o más clases concretas con el fin de entregar los objetos específicos.

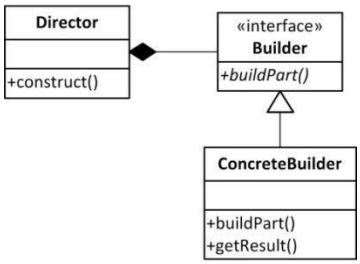
Se utiliza cuando

- La creación de objetos debe ser independiente del sistema de la utilización de ellos.
- Los sistemas deben ser capaces de utilizar múltiples familias de objetos.
- Las familias de los objetos se deben usar juntos.
- Las bibliotecas deben ser publicados sin exponer a los detalles de implementación.
- Clases de concreto deben ser disociados de los clientes.

Ejemplo

Los editores de correos permitirán edición en múltiples formatos, incluyendo texto plano, texto enriquecido y HTML. Dependiendo del formato utilizado, tendrán que crearse diversos objetos. Si el mensaje es texto plano entonces no puede haber un objeto corporal que representaba sólo texto sin formato y un objeto adjunto que sólo encripta los datos adjuntos en Base64. Si el mensaje es HTML, entonces el objeto cuerpo representaría texto codificado HTML y el objeto adjunto que permitiría una representación en línea y un accesorio estándar. Mediante la utilización de una fábrica de extracto de la creación, entonces podemos asegurar que los conjuntos de objetos apropiados se crean en base al estilo de correo electrónico que se está enviando.

Builder



Propósito

Permite la creación dinámica de objetos basado en algoritmos fácilmente intercambiables.

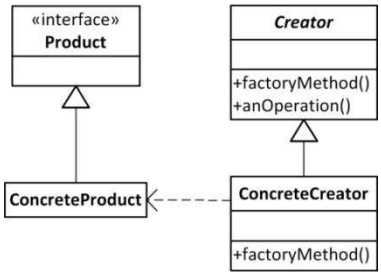
Se utiliza cuando

- Algoritmos de creación de objetos deben ser desconectados del sistema.
- Se requieren múltiples representaciones de algoritmos de creación.
- La adición de nueva funcionalidad de creación sin cambiar el código del núcleo es necesario.
- Se requiere un control de tiempo de ejecución sobre el proceso de creación.

Ejemplo

Una aplicación de transferencia de archivos podría utilizar diferentes protocolos para enviar archivos y el objeto de transferencia real que se creará será directamente dependiente del protocolo elegido. El uso de un generador se puede determinar el constructor de derecho de uso de una instancia del objeto correcto. Si el ajuste es FTP entonces el constructor de FTP se utiliza cuando se crea el objeto.

Factory Method



Propósito

Expone un método para la creación de objetos, lo que permite subclases para controlar el proceso de creación real.

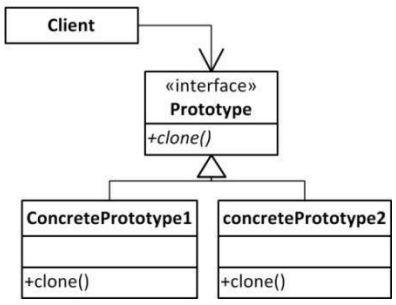
Se utiliza cuando

- Una clase no sabrá qué clases será necesario crear.
- Las subclases pueden especificar qué objetos deben ser creada.
- Clases de padres desean aplazar la creación de sus subclases.

Ejemplo

Muchas aplicaciones tienen algún tipo de usuario y la estructura del grupo de seguridad. Cuando la aplicación necesita para crear un usuario que será típicamente delegar la creación del usuario en varias implementaciones de usuario. El objeto de usuario padres se encargará de la mayoría de las operaciones para cada usuario, pero las subclases definirán el método de fábrica que se encarga de las distinciones en la creación de cada tipo de usuario. Un sistema puede tener objetos adminuser y StandardUser cada uno de los cuales extienden el objeto Usuario. El objeto AdminUser puede realizar algunas tareas adicionales para garantizar el acceso, mientras que el StandardUser puede hacer lo mismo para limitar el acceso.

Prototype



Propósito

Crear objetos basados en una plantilla de unos objetos existentes a través de la clonación.

Se utiliza cuando

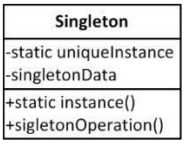
- Composición, la creación y la representación de los objetos deben ser desconectados de un sistema.
- Las clases que se creen se especifican en tiempo de ejecución.
- Un número limitado de combinaciones de estado existe en un objeto.
- Se requieren objetos o estructuras de objetos que son idénticos o muy parecidos a otros objetos o estructuras de objetos existentes.
- La creación inicial de cada objeto es una operación costosa.

Ejemplo

Cambio de procesamiento de motores a menudo requieren la búsqueda de muchos valores de configuración diferentes, haciendo que la inicialización del motor de un proceso relativamente caro. Cuando se necesitan varias instancias del motor, por ejemplo para importar datos de un modo multi-hilo, el costo de la inicialización de muchos motores es alto. Al utilizar el modelo de prototipo podemos asegurar que sólo una única copia del motor tiene que ser inicializado simplemente clonar el motor para crear un duplicado del objeto que ya está iniciado.

El beneficio adicional de esto es que los clones se pueden optimizar para incluir sólo los datos relevantes para su situación.

Sigleton



Propósito

Asegura que sólo una instancia de una clase se permite dentro de un sistema.

Se utiliza cuando

- Se requiere exactamente una instancia de una clase.
- Acceso controlado a un único objeto es necesario.

Ejemplo

La mayoría de los lenguajes proporcionan algún tipo de sistema o el medio ambiente objeto que permite el lenguaje para interactuar con el sistema

operativo nativo. Dado que la aplicación se está ejecutando físicamente en un solo sistema operativo sólo hay siempre una necesidad de una sola instancia de este objeto de sistema. El patrón se aplicaría en lenguaje en tiempo de ejecución para garantizar que sólo se crea una copia del objeto del sistema y asegurar sólo se permiten los procesos adecuados de acceso a la misma.

REFERENCIAS

[1] *Patterns in Java volume I, 2th edition, Mark Grand*

[2] *Design Patterns: Elements of Reusable Object-Oriented Software*