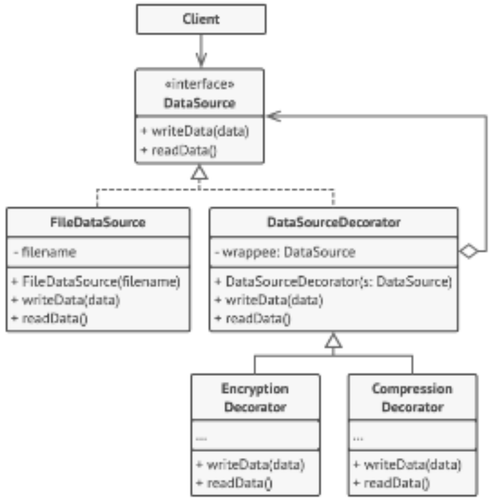
Gustavo Pérez Badilla - 12/05/23
2020084832

Caso 1:

Un jugador puede crear un personaje en la cual puede seleccionar las habilidades que este tiene. El personaje empieza con la habilidad de pelear cuerpo a cuerpo. El jugador deberá seleccionar otras habilidades que crea necesario para superar el siguiente nivel. Cada nivel se crea un nuevo personaje y tiene diferentes dificultades, por lo que, al completar un nivel, el jugador tendrá que seleccionar nuevamente las habilidades del personaje que le permitirá completar el siguiente nivel.

Patrón: Decorator

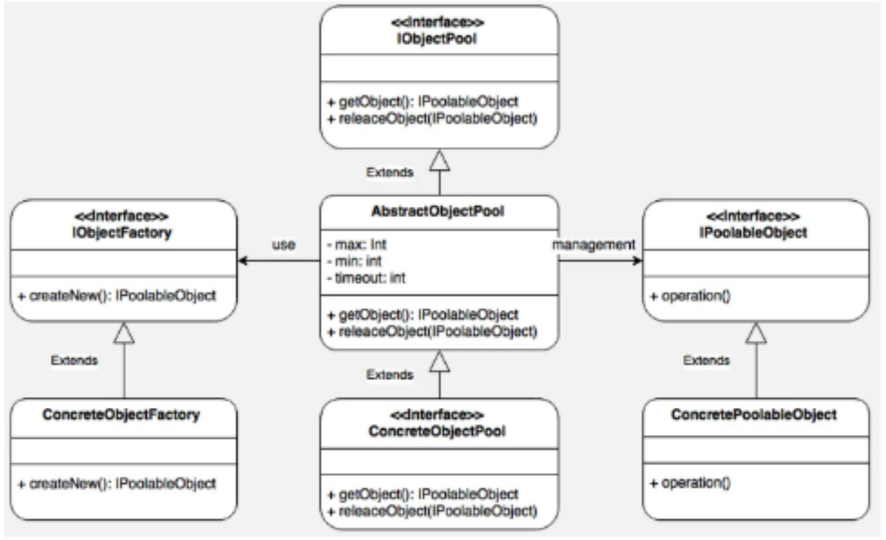


En este caso el personaje puede obtener diferentes habilidades, y este se reinicia en cada nivel, por lo que el personaje sería el objeto principal, y las habilidades serían decoradores.

Caso 2:

Un personaje en un videojuego dispone de una serie de drones que lo ayudan a combatir, esto lo puede hacer durante cualquier momento del juego. El problema es que los drones dentro del juego tienen mucho detalle y generan mucha carga gráfica cada que se genera uno nuevo. Además, los drones que no se utilizan o se termina su tiempo de uso, están en el campamento del jugador esperando ser utilizados.

Patrón: Object Pool

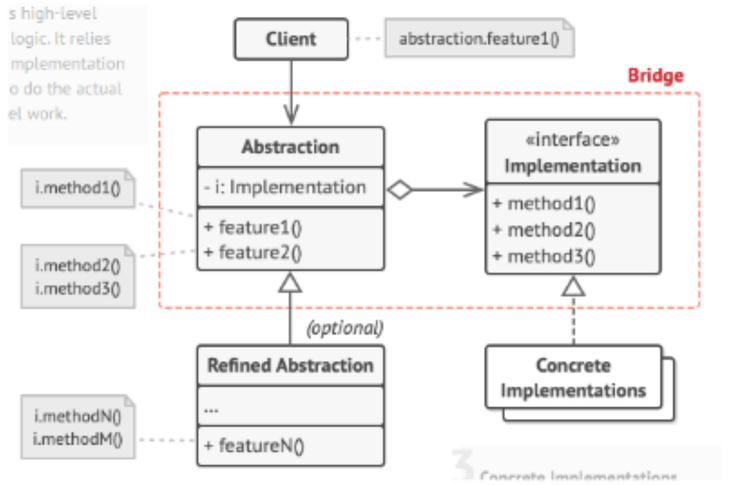


Estamos hablando de una cantidad de objetos que pueden generarse y removerse en cualquier momento, por lo que es mejor contar con un arreglo con todos los objetos y habilitarlos cuando son necesitados, en este caso serían los drones, y serían habilitados cuando el usuario lo desee.

Caso 1:

Si se estuviese desarrollando una aplicación de dibujo que debe funcionar con diferentes tipos de dispositivos de entrada, como ratones, lápices ópticos y pantallas táctiles. Cada dispositivo de entrada tiene su propia forma de capturar e implementar los eventos de dibujo, y queremos diseñar nuestra herramienta de dibujo para que pueda trabajar y dibujar con cualquier dispositivo de entrada sin tener que modificar el código de la aplicación para cada uno.

Patrón: Bridge

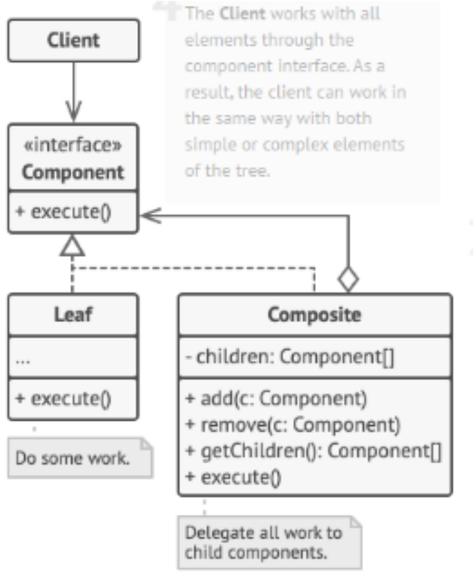


En este caso se puede hacer una conexión entre el dispositivo de entrada y las funcionalidades que este pueda ejecutar gracias a el tipo de dispositivo que es.

Caso 2:

Supongamos que estamos diseñando un sistema de navegación para un automóvil que debe mostrar información sobre diferentes rutas y puntos de interés al conductor. La información se presenta en forma de nodos de árbol, donde cada nodo representa un punto de interés o una ruta que el conductor puede seleccionar. Una ruta puede contener varios puntos de interés

Patrón: Composite



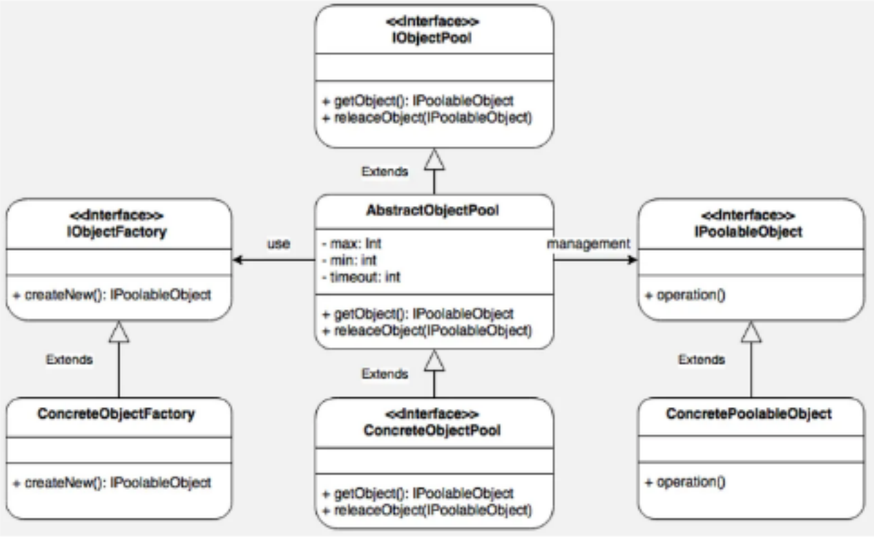
En este caso se van agregando puntos de importancia dentro de una ruta principal, por lo que estos puntos resultan ser nodos del arbol.

Gustavo Pérez Badilla - 17/05/23
2020084832

Caso 1:

Se está desarrollando un juego donde uno de los objetivos del personaje principal es cuidar de los distintos animales en una granja. Si los animales no están siendo cuidados correctamente tienen la peculiaridad de que pueden escapar de su encierro, por lo que el personaje principal deberá recuperarlos. Una vez un animal ha sido atrapado, este regresa a su establo correspondiente en la granja principal. Se debe considerar que el detalle gráfico de los animales en el juego representa un costo muy elevado, por lo que la reutilización de los animales representa un factor esencial en la aplicación.

Patrón: Object Pool

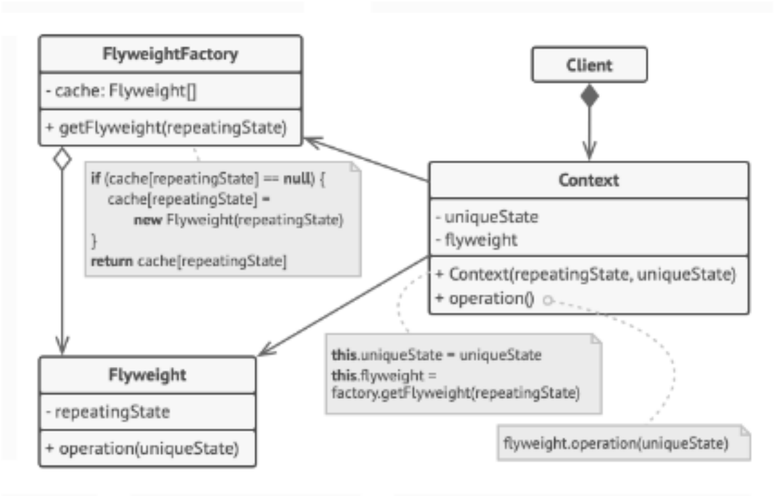


En este caso los objetos pueden ser reutilizados y se ahorrarían recursos al habilitar y deshabilitar los objetos dentro del modelo, En este caso los animales serían el objeto, y el Object Pool sería un arreglo con todos los mismos.

Caso 2:

Suponga que se está desarrollando un video juego en donde cada vez que se avanza de nivel los escenarios de ambientación del juego también cambian. Cada escenario contiene múltiples objetos (vegetación, rocas, edificios, etc) que pueden tener características comunes, como la apariencia visual, sin embargo, se espera poder ubicarlos en distintas posiciones a lo largo del mapa del nivel. La creación individual de los objetos consume mucha memoria en el video juego por lo que se espera que el patrón a implementar ayude a la reducción de costos.

Patrón: Flyweight



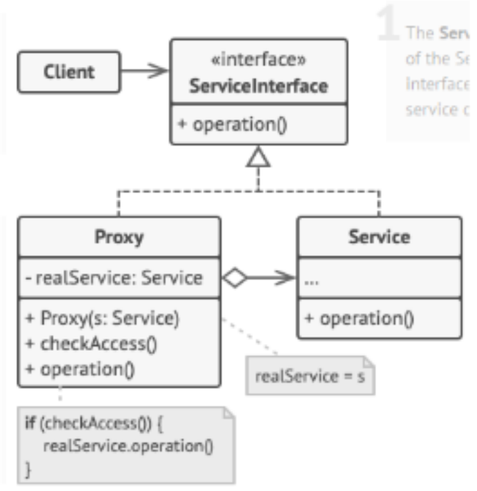
Se emplea el Flyweight debido a que los objetos comparten ciertos atributos como las texturas, siendo estos empleados dentro del contexto, y los atributos propios como la ubicacion, serían propios de cada objeto, ahorrando espacio en memoria.

Gustavo Pérez Badilla - 17/05/23
2020084832

Caso 1:

Estás desarrollando una aplicación de transmisión de videos en la que los usuarios pueden ver películas y series en línea. La aplicación obtiene los contenidos de un servidor. Por motivos de seguridad, quieres restringir el acceso, únicamente a usuarios que estén registrados en la aplicación, además de implementar una forma de almacenar el contenido visto por el usuario.

Patrón: Proxy

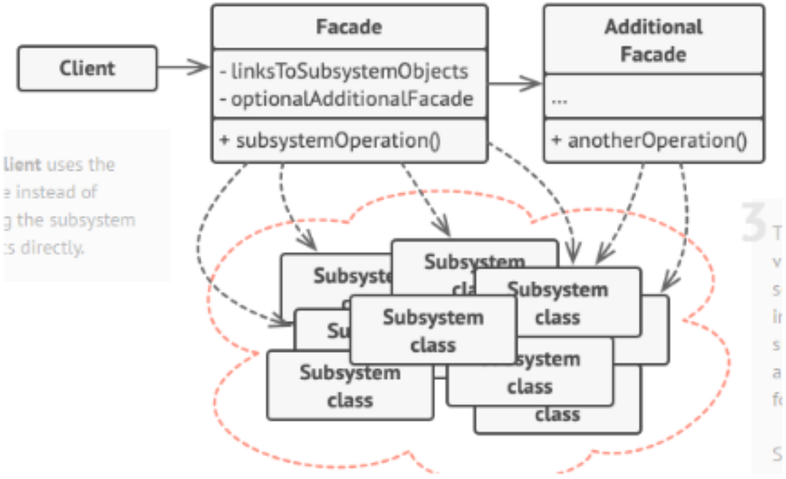


En este caso se hace la validación de la ubicación con el Proxy, de forma que el contenido que se despliegue sea el correspondiente según la posición del usuario.

Caso 2:

Estás desarrollando un sistema de compras en línea para una tienda. El sistema debe interactuar con varios componentes, tales como el inventario, el proceso de pago, la gestión de envíos y el control de usuarios. Las principales tareas que debe poder realizar el sistema son: verificar la disponibilidad de un producto, procesar el pago del cliente, registrar la información de envío al usuario y la creación y autenticación de cuentas de usuarios.

Patrón: Facade



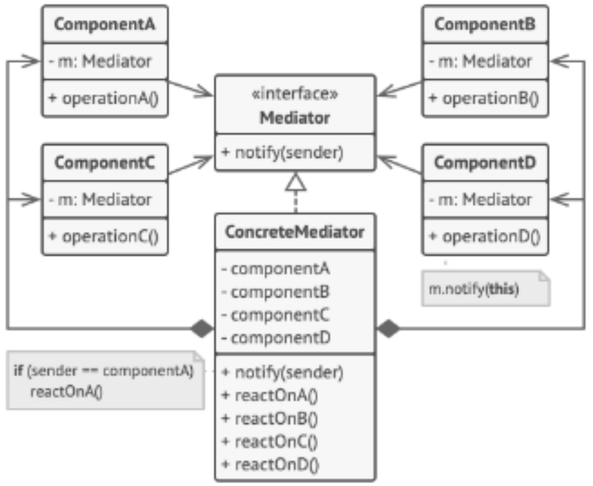
Se emplea el Flyweight debido a que los objetos comparten ciertos atributos como las texturas, siendo estos empleados dentro del contexto, y los atributos propios como la ubicacion, serían propios de cada objeto, ahorrando espacio en memoria.

Gustavo Pérez Badilla - 24/05/23
2020084832

Caso 1:

Sistema de Gestión de Aeropuertos
Imagina un sistema de gestión de aeropuertos que maneja diferentes componentes, como vuelos, pasajeros, aerolíneas y controladores de tráfico aéreo. El sistema necesita coordinar la comunicación entre estos componentes para garantizar un funcionamiento eficiente y seguro del aeropuerto. A medida que el aeropuerto crece y se añaden más aerolíneas y vuelos, la comunicación entre los diferentes componentes se vuelve cada vez más compleja. Además, los controladores de tráfico aéreo deben interactuar con múltiples vuelos y aerolíneas al mismo tiempo, lo que dificulta su gestión y mantenimiento.

Patrón: Mediator

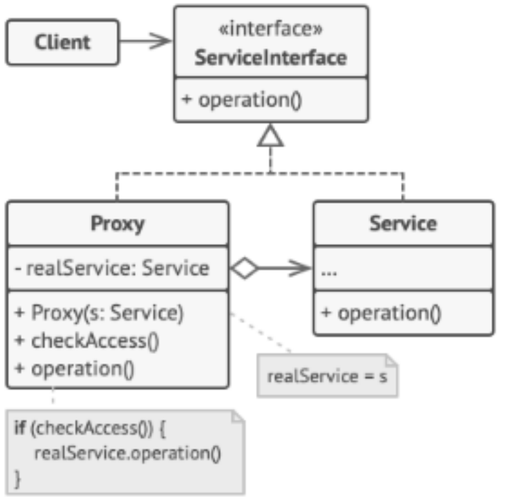


Cada componente (vuelos, pasajeros, aerolineas, entre otros.) pasa por medio del Mediator (sistema), el cual busca garantizar la comunicación entre los componentes, y los componentes se comunican por medio de este sistema, no entre ellos.

Caso 2:

Acceso de Archivos
Imaginemos una empresa que tiene múltiples sucursales distribuidas geográficamente en diferentes ciudades. Cada sucursal tiene empleados que necesitan acceder a archivos y documentos almacenados en un servidor centralizado ubicado en la sede principal de la empresa. Sin embargo, debido a las limitaciones de ancho de banda y la distancia física entre las sucursales y el servidor central, el acceso directo a los archivos remotos puede resultar lento e ineficiente.

Patrón: Proxy



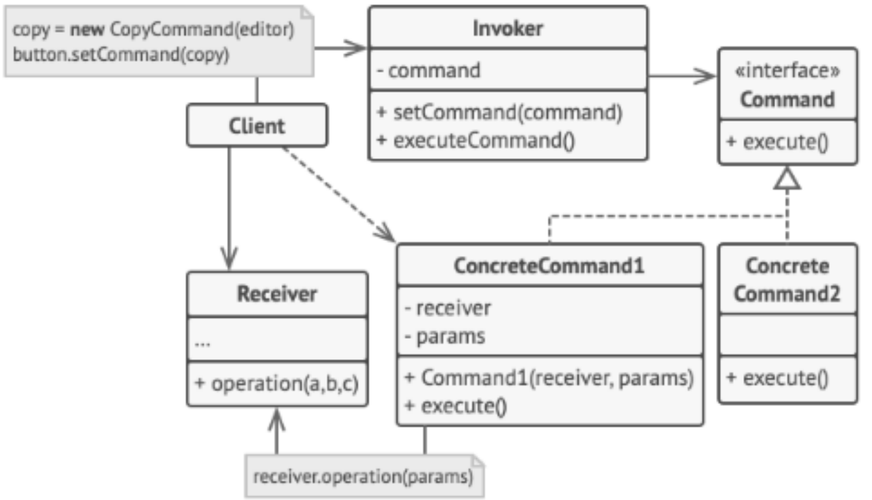
Cada usuario pasa por un filtro (interfaz) que procesa los datos en el proxy, y luego el servicio envía los datos correspondientes según la región.

Gustavo Pérez Badilla - 24/05/23
2020084832

Caso 1:

El diseño de un sistema de control de luces de un automóvil. El objetivo es permitir al conductor encender, apagar y ajustar diferentes luces, como los faros delanteros, las luces interiores y las luces de señalización.

Patrón: Command

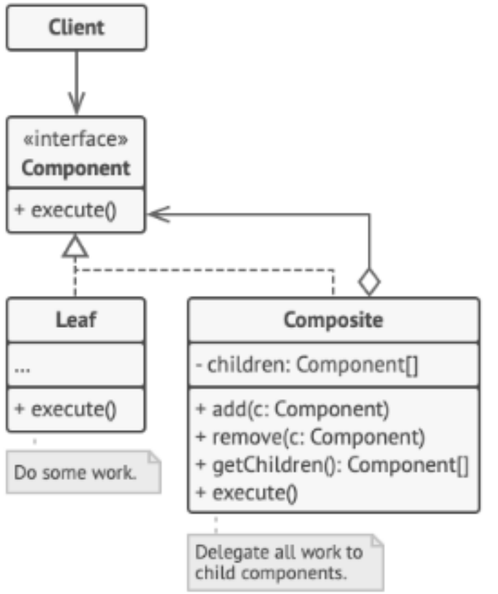


El Invoker corresponde al conductor quien se comunica con la interfaz para realizar los comandos concretos (acciones del carro como encender, apagar, ajustar las luces, entre otros).

Caso 2:

Crear un sistema de pedidos de productos. Los pedidos pueden contener productos sencillos sin envolver, así como cajas llenas de productos... y otras cajas.

Patrón: Composite



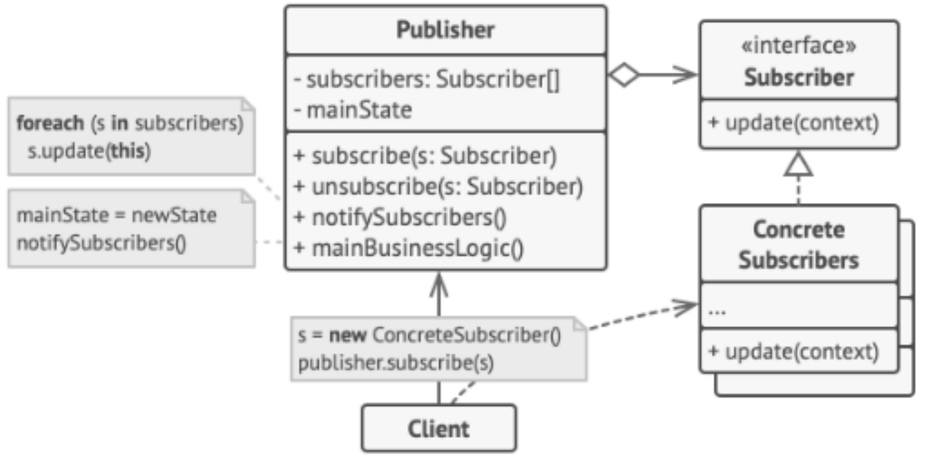
Los productos corresponden a los nodos hijos que almacenan la información necesaria, y un pedido corresponde al componente inicial.

Gustavo Pérez Badilla - 26/05/23
2020084832

Caso 1:

La fábrica de Galletas Paquita cuenta con un único horno que al terminar de hornear la tanda de galletas activa las bandas que transportan las galletas a las líneas de inspección. Además, se le avisa a los inspectores que las galletas están listas para que inicien su trabajo.

Patrón: Observer

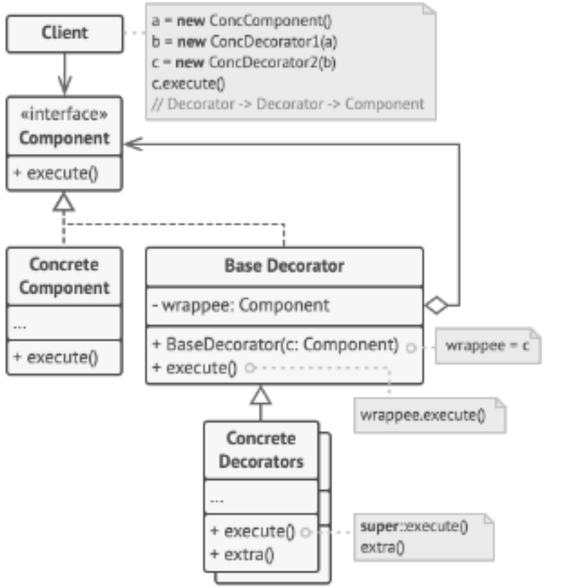


En este caso el Publisher seria el encargado de la creación de las galletas, mientras que los Concrete Subscribers corresponden a

Caso 2:

Galletas Paquita hace todas sus galletas con una masa base. A la base se le agregan ingredientes como chispas de chocolate, coco, nueces y frutos secos para crear los sabores que ofrecen. A una base se le pueden agregar varios tipos de ingredientes.

Patrón: Decorator



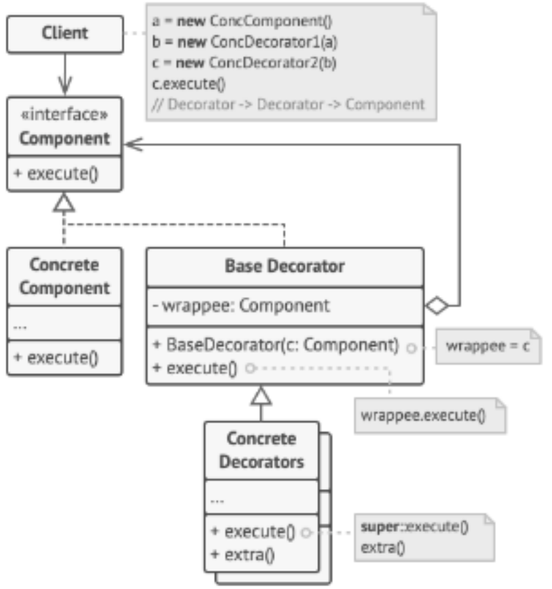
En este caso el componente corresponde a las galletas, y los decoradores reflejarían los topings que se le desee agregar a la galleta.

Gustavo Pérez Badilla - 26/05/23
2020084832

Caso 1:

En un sistema de ventas online de una tienda de ropa, se desea implementar una funcionalidad que permita aplicar diferentes tipos de descuentos a los productos durante el proceso de compra. Cada descuento puede ser de naturaleza distinta, como descuento por cantidad, descuento por membresía o descuento por temporada.

Patrón: Decorator

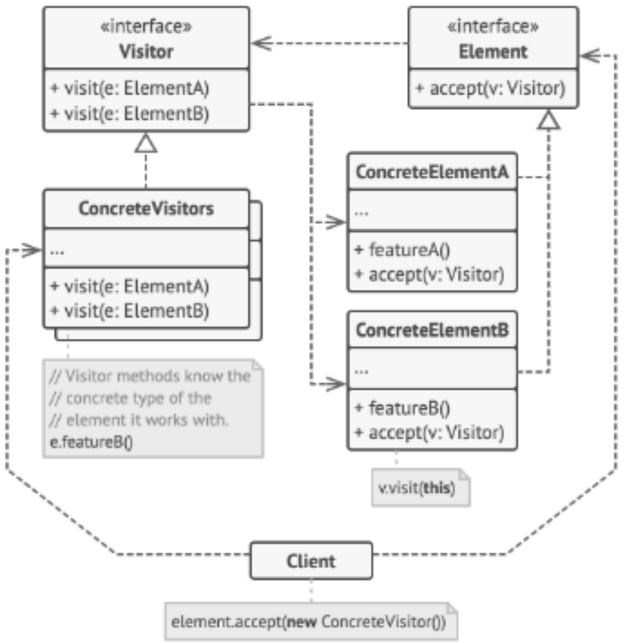


En este caso los decoradores juegan el papel del descuento que se les aplica a los productos, mientras que los componentes corresponden a los productos a los que se les agrega el descuento.

Caso 2:

En un museo se desea implementar un sistema en el que se tienen diferentes tipos de obras de arte, como pinturas, esculturas y fotografías, y se desean realizar diversas operaciones en ellas, como calcular su valor, obtener una descripción detallada y calcular métricas específicas de cada tipo de obra

Patrón: Visitor



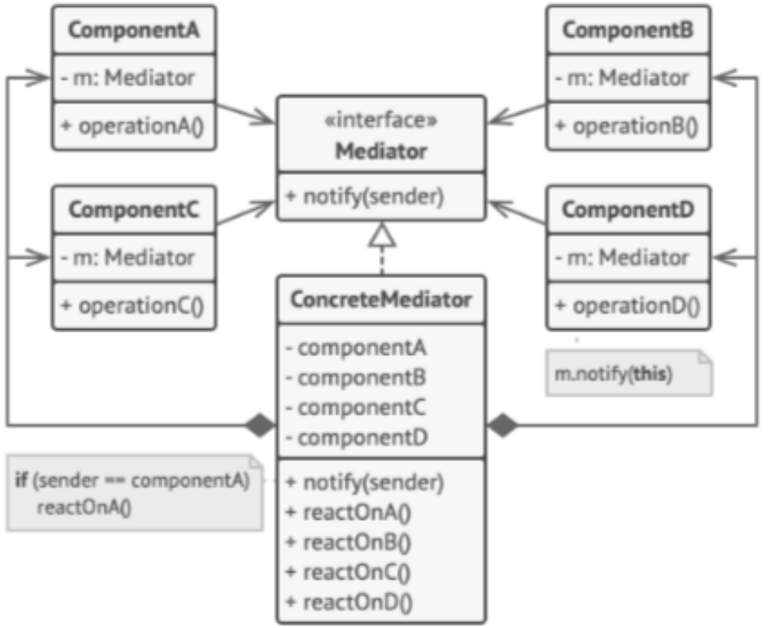
El Concrete Element corresponde a las obras de arte, mientras que las operaciones que se desean realizar corresponden a los Concrete Visitors.

Gustavo Pérez Badilla - 31/05/23
2020084832

Caso 1:

Suponga que está desarrollando un sistema de chat en línea donde múltiples usuarios pueden comunicarse entre sí en un mismo chat. Cada usuario tiene su propia interfaz de chat y puede enviar mensajes a los otros usuarios

Patrón: Mediator

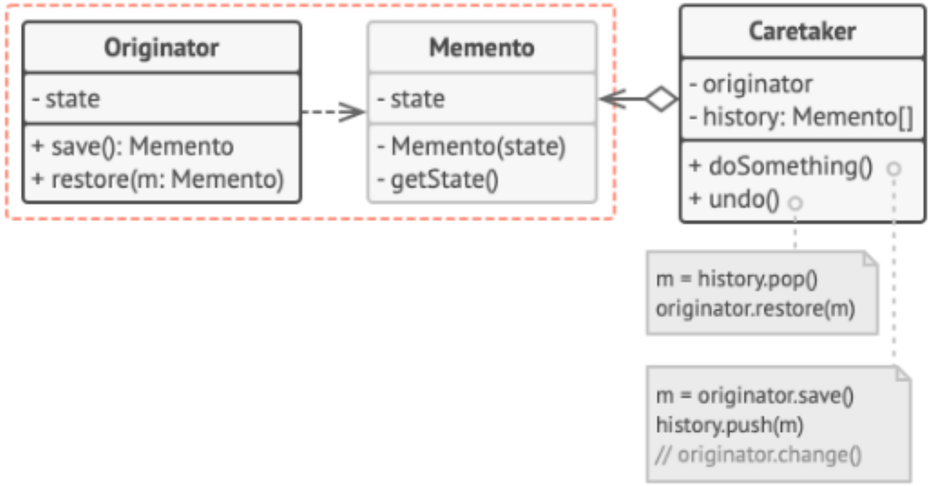


En este caso el grupo del chat corresponde al Mediator, donde cada componente es un usuario.

Caso 2:

Para la implementación de un videojuego, se desea agregar la funcionalidad de guardar y cargar partidas. Debería permitir guardar el estado actual del juego en un momento y luego restaurarlo en un momento posterior.

Patrón: Memento



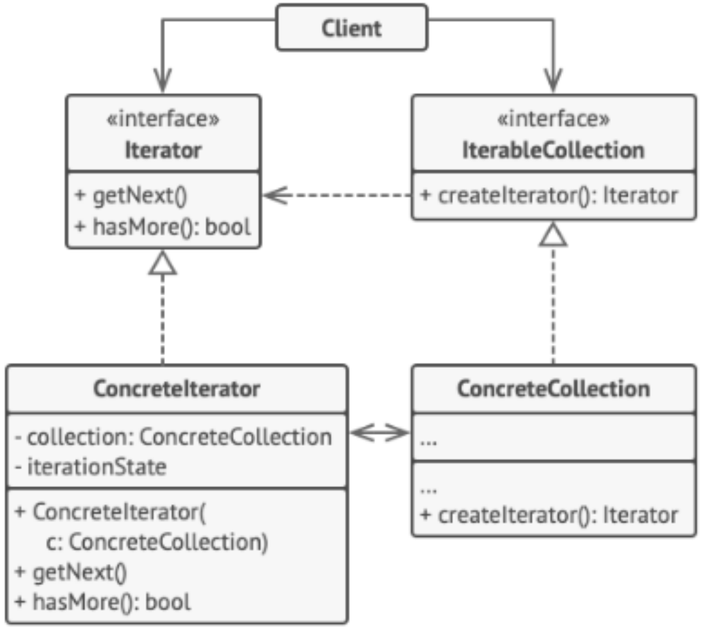
El Originator sería el juego que busca la funcionalidad de guardar y cargar partidas, el Memento corresponde al estado en el que se encuentra el juego, y el Caretaker el responsable de almacenar estos Mementos.

Gustavo Pérez Badilla - 31/05/23
2020084832

Caso 1:

Recorriendo una lista de tareas pendientes Imagina que estás desarrollando una aplicación de gestión de tareas y necesitas implementar una funcionalidad para recorrer una lista de tareas pendientes de un usuario. Deseas permitir que el usuario pueda ver las tareas de manera secuencial, obteniendo una tarea a la vez.

Patrón: Iterator

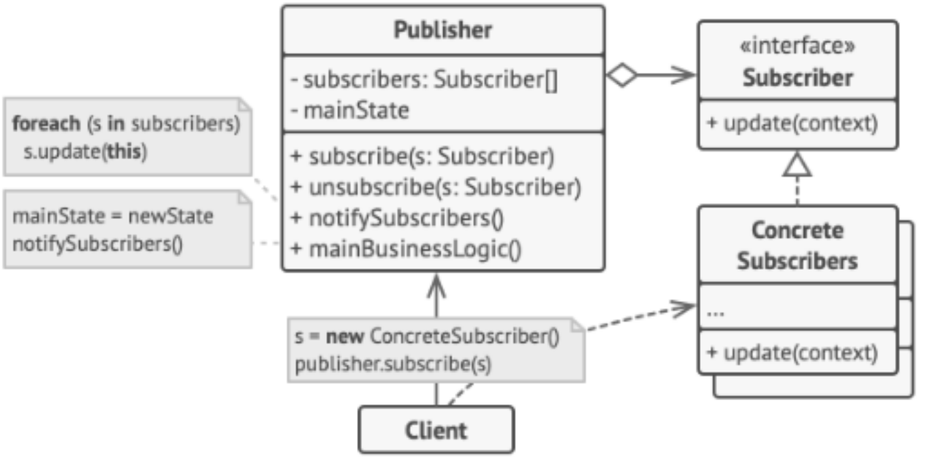


En este caso el iterador es el responsable de mostrar los datos de cada tarea, los Concrete Iterator son los responsables de recorrer la lista de tareas.

Caso 2:

Tienes un sistema de notificaciones en una aplicación de chat. Deseas implementar una funcionalidad en la que los usuarios puedan suscribirse a determinados canales de chat y recibir notificaciones cuando se publiquen nuevos mensajes en esos canales.

Patrón: Observer

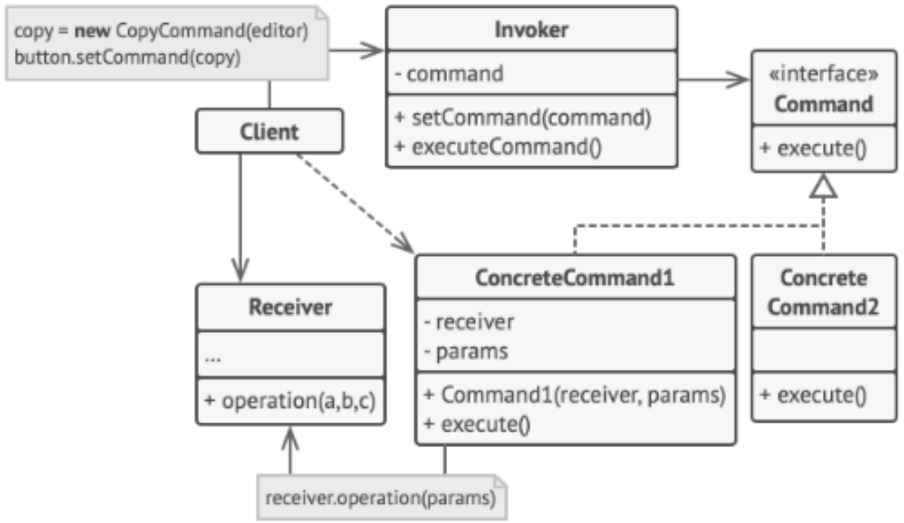


El Observer ya que los usuarios corresponden a los Publisher y cada uno cuenta con sus suscriptores, es decir los destinarios.

Caso 1:

Una compañía de juguetes para niños desea vender un robot que pueda realizar diferentes acciones como caminar, correr, bailar y saltar. La compañía desea que el niño pueda utilizar un único dispositivo que le permita accionar el robot a distancia.

Patrón: Command

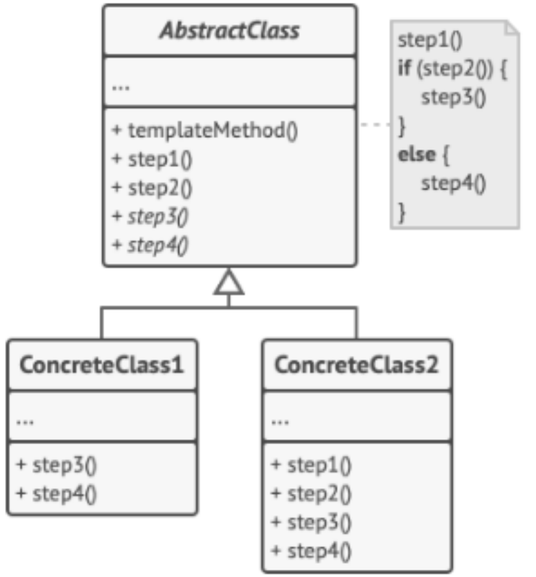


El Invoker corresponde al dispositivo desde el cual se puede controlar al robot, el receiver corresponde al robot que recibe las instrucciones, el Command corresponde a la interfaz, y los Concrete Command son cada una de las instricciones como caminar, correr, bailar, saltar, entre otros.

Caso 2:

Suponga que va a crear un videojuego sobre una pizzería, donde los pasos para preparar las pizzas son: Hacer la masa, preparar la salsa, poner el resto de los ingredientes, meter al horno y, por último, servirla. Existen diferentes tipos de pizza, sin embargo, todas requieren de los pasos mencionados anteriormente y se busca la manera más eficiente de manejar la línea de producción.

Patrón: Template method

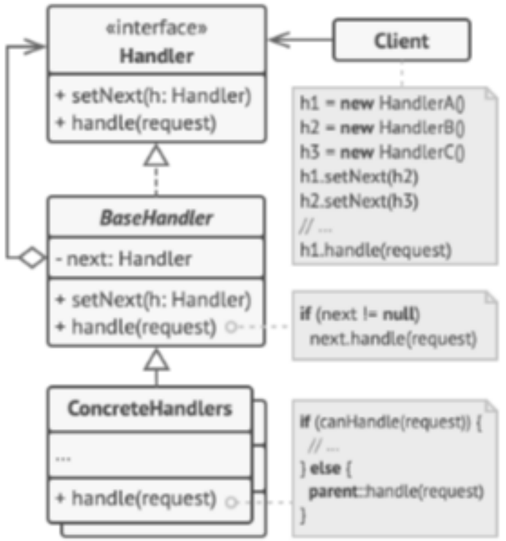


En este caso la clase abstracta corresponde a hacer la pizza, y cada clase concreta es cada sub proceso o tarea para realizar la pizza.

Caso 1:

En un sistema de soporte técnico, los usuarios pueden enviar solicitudes de asistencia para diferentes problemas. Existen diferentes tipos de técnicos de soporte, como Técnico Nivel 1, Técnico Nivel 2 y Técnico Nivel 3, cada uno con diferentes niveles de experiencia. Se necesita un mecanismo para resilver las solicitudes de asistencia de manera eficiente y escalable, asignándolas al técnico adecuado según la complejidad del problema.

Patrón: Chains of responsability

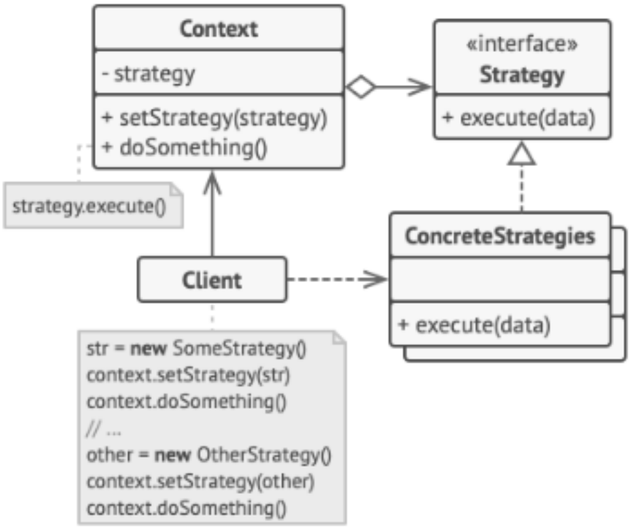


En este caso, cada técnico corresponde a cada uno de los Handlers, donde si el de primer nivel no logra solucionar el problema, pasa al técnicc de segundo nivel, y así sucesivamente. De esta forma se van solucionando los problemas de acuerdo al nivel de complejidad.

Caso 2:

En un sistema de envío de paquetes, los clientes pueden elegir entre diferentes opciones de entrega, como entrega estándar, entrega rápida y entrega exprés. Cada opción de entrega tiene diferentes costos y tiempos de entrega. Se requiere un mecanismo para determinar de manera flexible la opción de entrega según las preferencias y necesidades del cliente.

Patrón: Strategy

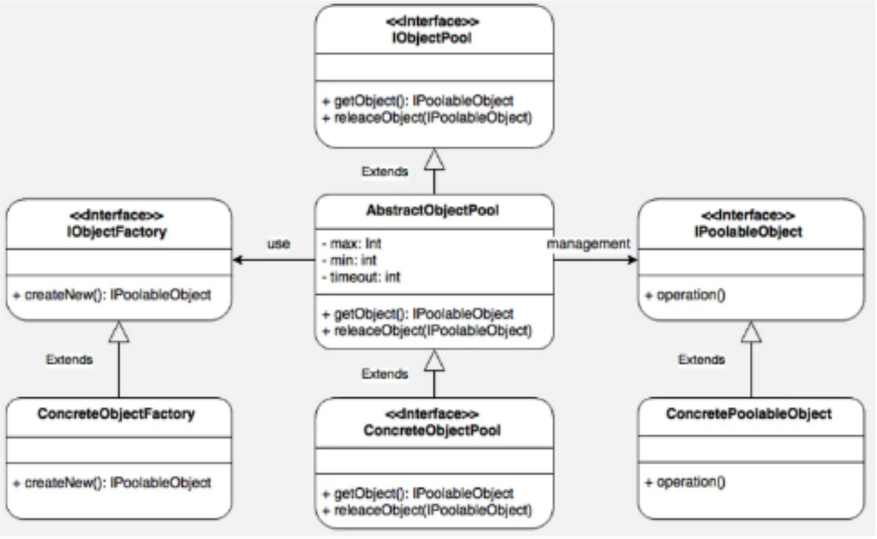


En este caso el usuario escoge el método que mejor se acople a su necesidad, en este caso las opciones de entrega corresponden a las estrategias concretas, mientras que el contexto corresponde al sistema de envío de paquetes.

Caso 1:

Existe un sistema de gestión de alquiler para automóviles. Los automóviles disponibles que están guardados en el garaje son utilizados por diferentes conductores u después de ser utilizados, vuelven al garaje para ser almacenados hasta que sean necesitados nuevamente.

Patrón: Object pool

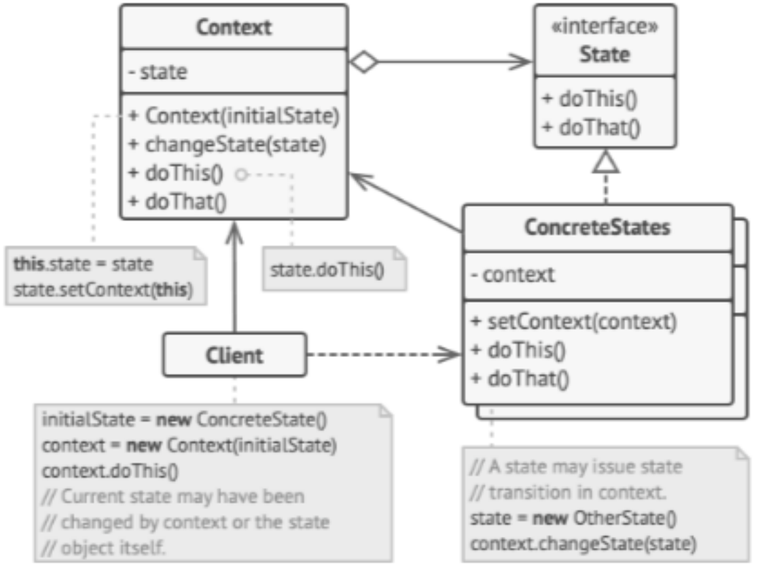


En este caso los automóviles corresponden al objeto contenido dentro del Pool, mientras que el sistema de gestión de alquiler de automóviles sería el Object Pool. Ya que los vehículos son devueltos al sistema, estos se pueden determinar como liberados u ocupados, de la misma manera que lo estipula este patrón.

Caso 2:

Existe un editor de texto el cual tiene 3 modos. Dependiendo del modo puede insertar, seleccionar o cambiar de estado. En el modo insertar puede, como dice el nombre, insertar texto en el documento. En el modo visual lo que se puede hacer es seleccionar bloques de texto haciendo que se pueda borrar, copiar o remplazar el bloque. En el modo normal lo único que puede hacer es cambiar de estado. Para cambiar de estado debe de presionar una tecla específica.

Patrón: State



En este caso el editor de texto corresponde al contexto, quien se encarga de determinar el estado en el que se encuentra. Los estados en concreto corresponden a cada una de las opciones que integra el programa (insertar, seleccionar, cambiar, entre otros).