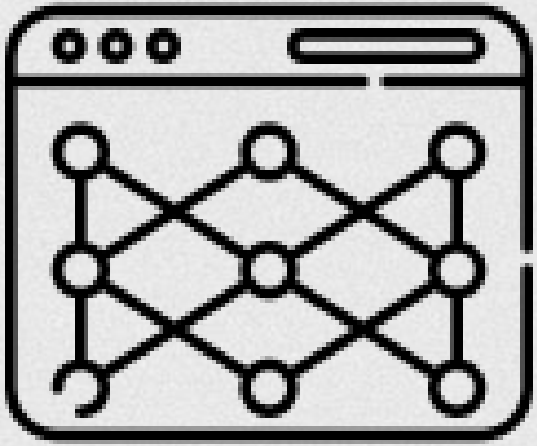
Patrones de Comportamiento

Command,
Chain of responsibility, Interpreter e iterator



Integrantes:
Cristian Alvias, Enson Chang
Bryan González, Eunice Morán y
Eric Vásquez

TABLA DE CONTENIDO

01

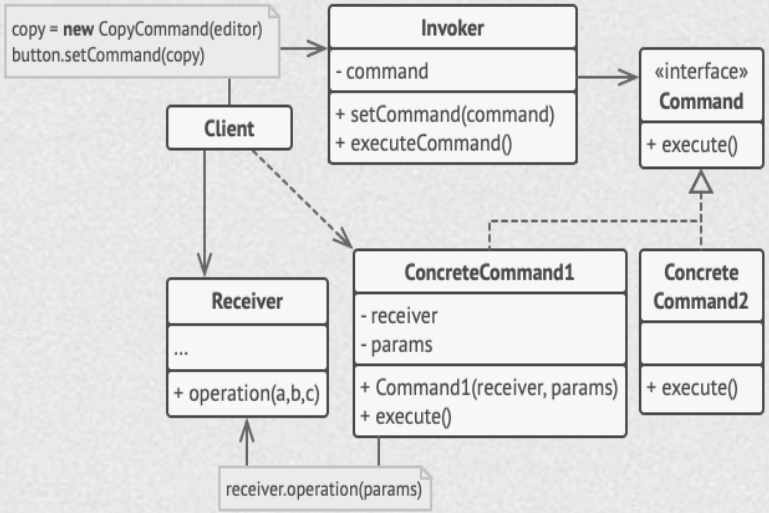
COMMAND

01 COMMAND

Proposito y Estructura

Es un patrón de diseño de comportamiento que convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. Esta transformación permite parametrizar los métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y soportar operaciones que no se pueden realizar.

Este enfoque promueve la separación de preocupaciones al **s**separar lo que llama a la acción de las cosas que saben cómo hacerlo.



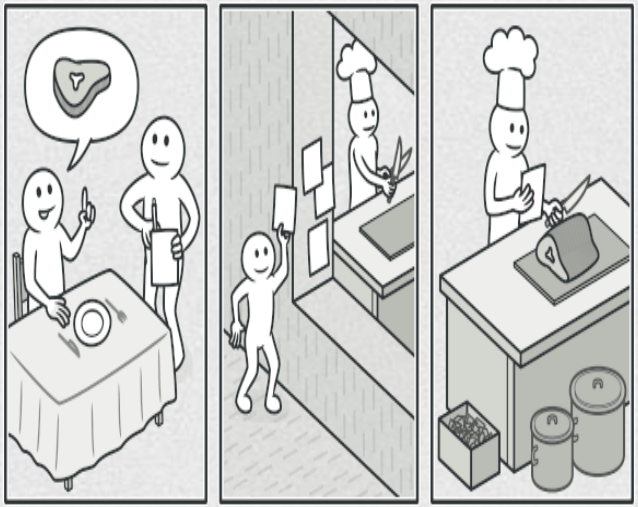
Su estructura está compuesta de la siguiente manera:

1. **Comando:** Define una interfaz común para todos los comandos usando el método execute(). Cada comando directo usa esta estructura y proporciona su propia lógica de ejecución.
2. **Invocador:** Es el encargado de llamar a los pedidos. Acepta un comando específico en un método y llama al método execute() cuando es necesario.
3. **Anfitrión:** Este es el objeto que hace el trabajo real cuando se ejecuta el comando. El comando hace que el host complete la solicitud.
4. **Cliente:** Crea un objeto de comando, lo asocia con el receptor apropiado y lo pasa a la persona que llama para su ejecución.

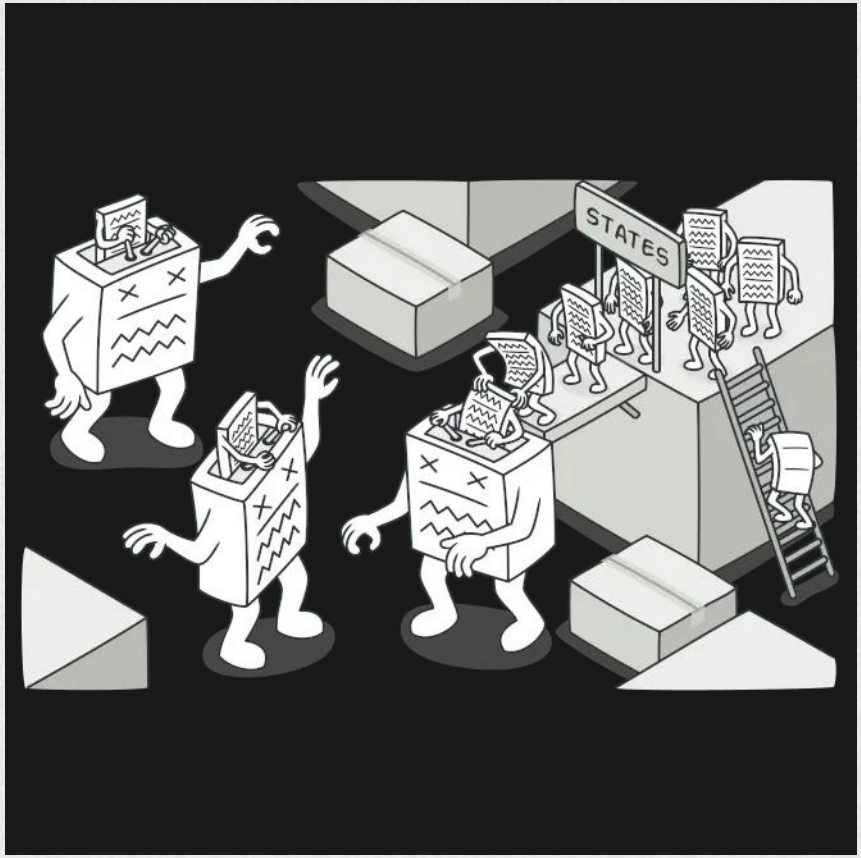
01 COMMAND

Ejemplo

- Tras un largo paseo por la ciudad, entras en un buen restaurante y te sientas a una mesa junto a la ventana. Un amable camarero se acerca y toma tu pedido rápidamente, apuntándolo en un papel. El camarero se va a la cocina y pega el pedido a la pared. Al cabo de un rato, el pedido llega al chef, que lo lee y prepara la comida. El cocinero coloca la comida en una bandeja junto al pedido. El camarero descubre la bandeja, comprueba el pedido para asegurarse de que todo está como lo querías, y lo lleva todo a tu mesa.



- El pedido en papel hace la función de un comando. Permanece en una cola hasta que el chef está listo para servirlo. Este pedido contiene toda la información relevante necesaria para preparar la comida. Permite al chef empezar a cocinar de inmediato, en lugar de tener que correr de un lado a otro aclarando los detalles del pedido directamente contigo.



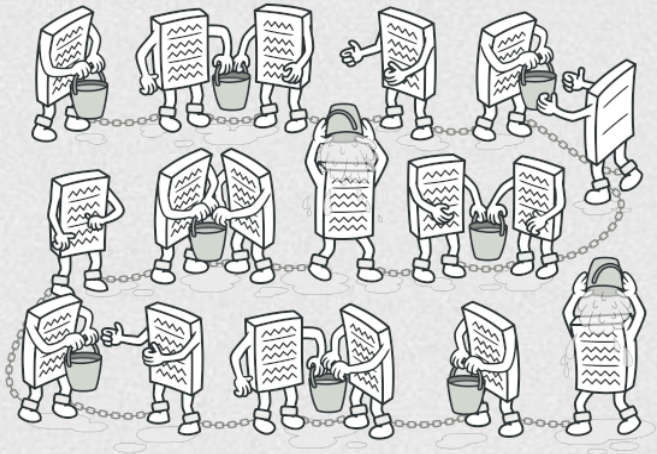
02

CHAIN OF RESPONSABILITY

02 CHAIN OF RESPONSABILITY

Permite establecer una cadena de objetos receptores que procesan una solicitud. Cada objeto puede manejar la solicitud o pasarla al siguiente objeto. El objetivo es desacoplar el remitente de los receptores y permitirles procesar la solicitud de forma independiente, formando una cadena completa de objetos enlazados.

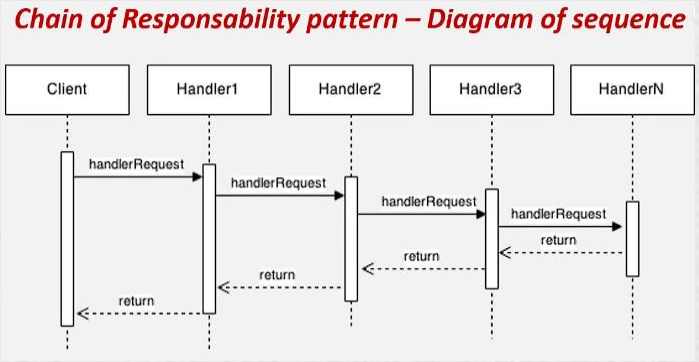
El patrón de cadena de responsabilidad permite que una solicitud se pase secuencialmente entre objetos hasta que uno de ellos la maneje, brindando flexibilidad al sistema y evitando afectar al remitente de la solicitud.



02 CHAIN OF RESPONSABILITY

Flujo de trabajo

- **Definición de la cadena:** Se crea una cadena de objetos receptores. Cada objeto en la cadena representa un paso o una etapa en el proceso de manejo de la solicitud.
- **Enlace entre los objetos:** Cada objeto en la cadena tiene una referencia al siguiente objeto en la secuencia. Esto establece el orden en el que se procesarán las solicitudes.
- **Recepción de la solicitud:** La solicitud se envía inicialmente al primer objeto en la cadena.
- **Verificación de la capacidad de manejo:** Cada objeto en la cadena verifica si puede manejar la solicitud. Si puede hacerlo, procesa la solicitud y termina el flujo.
- **Paso de la solicitud:** Si el objeto actual no puede manejar la solicitud, la pasa al siguiente objeto en la cadena. Esto continúa hasta que se encuentra un objeto capaz de manejar la solicitud o hasta que se llega al final de la cadena.
- **Finalización del flujo:** Una vez que la solicitud ha sido procesada, el flujo de trabajo termina.



Este patrón es especialmente útil cuando se requiere un procesamiento flexible de solicitudes con múltiples niveles de responsabilidad o cuando se desconoce de antemano qué objeto será capaz de manejar la solicitud. También ayuda a evitar el acoplamiento entre el emisor de la solicitud y los objetos receptores, ya que no se necesita conocer específicamente quién manejará la solicitud en cada paso.

02 CHAIN OF RESPONSABILITY

Ejemplo

Un sistema de aprobación de solicitudes de vacaciones para una empresa. Tienes tres niveles de aprobación: el supervisor, el gerente y el director. Cada nivel tiene una capacidad de aprobación diferente y la solicitud debe pasar por cada nivel en orden.

Clasificar las cadenas:

Supervisor: representa al supervisor de un empleado y tiene un límite de aprobación de hasta 5 días de vacaciones.

Gerente: representa al gerente de un departamento y tiene un límite de aprobación de hasta 10 días de vacaciones.

Director: representa al director general de la empresa y no tiene límite de aprobación.

Enlace entre los objetos: Cada objeto en la cadena tiene una referencia al siguiente objeto en la secuencia. Por ejemplo, el supervisor tiene una referencia al gerente, y el gerente tiene una referencia al director.

Recepción de la solicitud: Una solicitud de vacaciones llega al supervisor.

Verificación de la capacidad de manejo: El supervisor verifica si puede aprobar la solicitud. Si la cantidad de días de vacaciones solicitados es igual o inferior a 5, el supervisor aprueba la solicitud y se finaliza el flujo. De lo contrario, pasa la solicitud al siguiente nivel, que es el gerente.

Paso de la solicitud: El gerente recibe la solicitud y realiza la misma verificación. Si puede aprobar la solicitud (hasta 10 días de vacaciones), lo hace y finaliza el flujo. De lo contrario, la pasa al siguiente nivel, que es el director.

Finalización del flujo: El director recibe la solicitud y, al no tener un límite de aprobación, aprueba cualquier cantidad de días de vacaciones solicitados. El flujo de la cadena de responsabilidad llega a su fin.

03

INTERPRETER

03 INTERPRETER

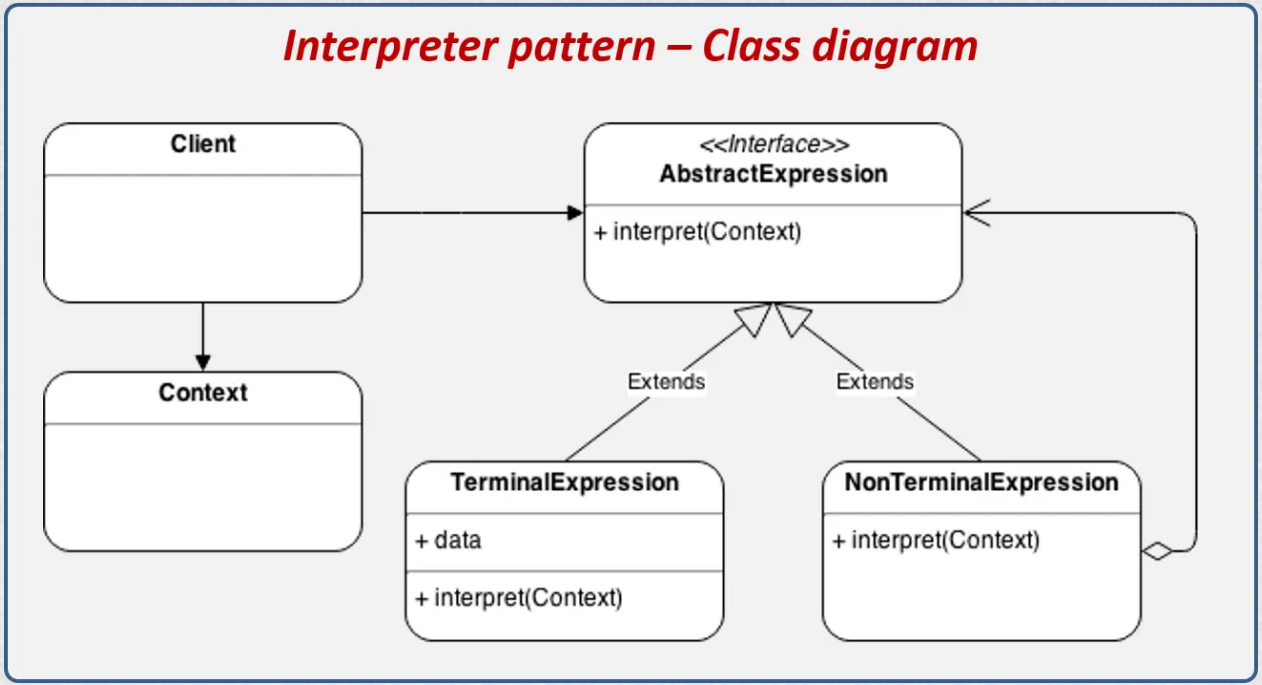
Es utilizado para evaluar un lenguaje definido como Expresiones, este patrón nos permite interpretar un lenguaje como *Java, C#, SQL* o incluso un lenguaje inventado por nosotros y darnos una respuesta tras evaluar dicho lenguaje.

Combinar técnicas de programación orientada a objetos avanzada.

Se compone de las siguientes clases:

- Cliente: Actor que dispara la ejecución del interpreter.
- Expresión: Define una interfaz común para todas las expresiones del lenguaje.
- Expresión Terminal: Representa una expresión básica. Esta interpreta y evalúa la expresión.
- Expresión No Terminal: Representa una expresión compuesta. Esta interpreta y evalúa una combinación de expresiones terminales y no terminales.
- Contexto: Contiene la información necesaria para interpretar y evaluar las expresiones.

03 INTERPRETER



Estructura del Patrón Interpreter

03 INTERPRETER

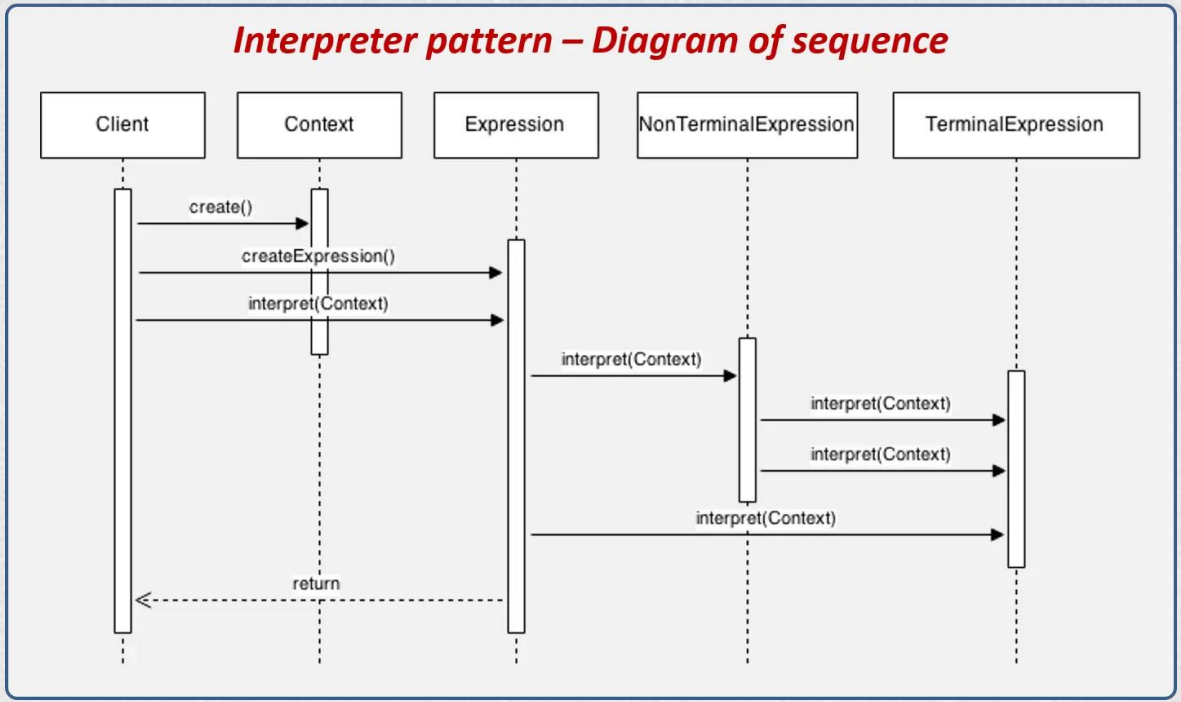


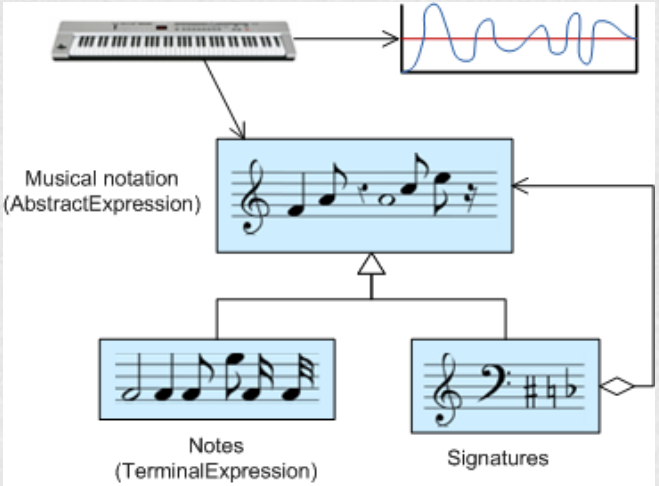
Diagrama de Secuencia del Patrón Interpreter

-El ***cliente*** crea el *contexto* para la ejecución.
-El ***cliente*** crea u obtiene la expresión a evaluar.
-El ***cliente*** solicita la interpretación de la expresión al *interpreter* y le envía el ***contexto***.
-La ***Expresión*** manda llamar a las ***Expresiones No Terminales***.
-La ***Expresión No Terminal*** manda llamar a todas las ***Expresiones Terminales***.
-La ***Expresión Raíz*** solicita la interpretación de una ***Expresión Terminal***.
-La ***expresión*** se evalúa por completo y se tiene un resultado de todas las *expresiones.*

03 INTERPRETER

Ejemplo 1

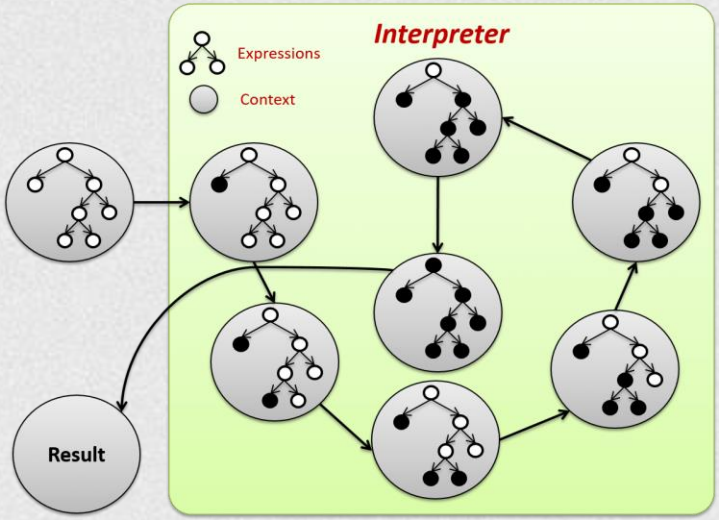
El patrón *Interpreter* define una representación gramatical para un lenguaje y un interprete para comprender esa gramática. Los músicos son ejemplos de *Interpreter*. El tono de un sonido y su duración puede ser representado en la notación musical de un pentagrama. Esta notación proporciona el lenguaje de la música. Los músicos que tocan la música de la partitura son capaces de reproducir el tono original y la duración de cada sonido representado.



03 INTERPRETER

Ejemplo 2

Mediante la implementación del patrón Interpreter se crea una aplicación que interprete comandos SQL para realizar consultas sobre un archivo de Excel, como si este se tratara de una base de datos relacional, en donde cada hoja será vista como una tabla y las columnas de la hoja como columnas de la tabla. Para esto se construye nuestra propia estructura de clases para representar el Lenguaje SQL, para finalmente ser interpretadas y nos arroje un resultado.



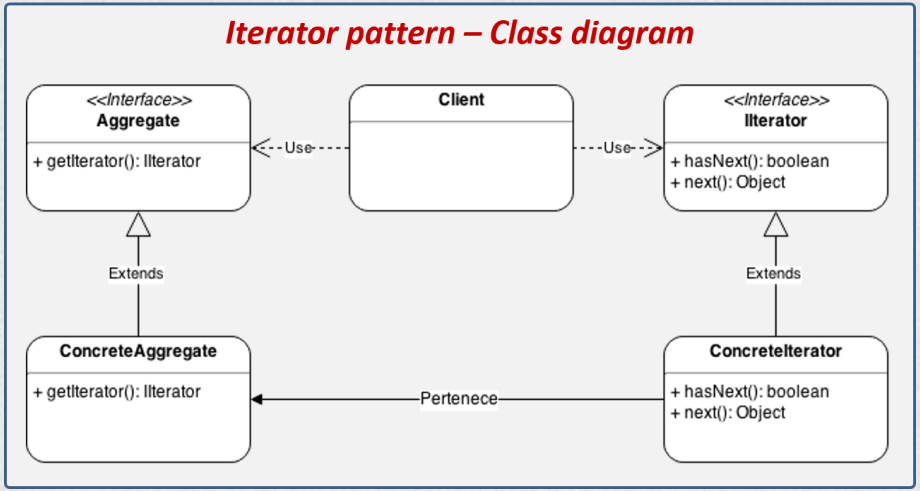
04

ITERATOR

04 ITERATOR

Este patrón de diseño permite recorrer una estructura de datos sin que sea necesario conocer la estructura interna de la misma.

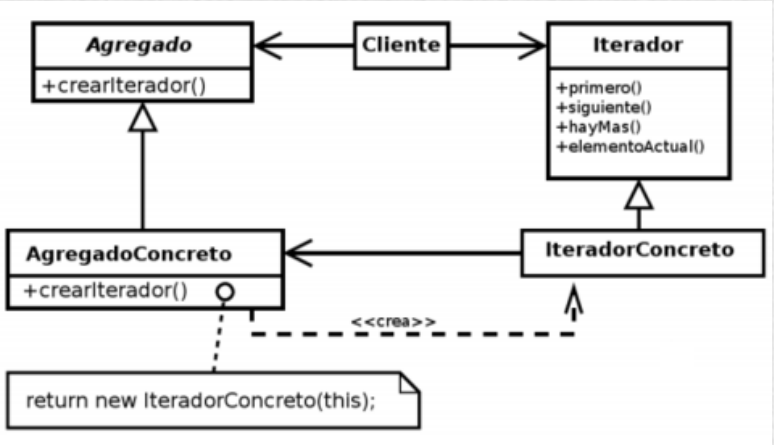
Es especialmente útil cuando trabajamos con estructuras de datos complejas, ya que nos permite recorrer sus elementos mediante un Iterador, el Iterador es una interface que proporciona los métodos necesarios para recorrer los elementos de la estructura de datos.



04 ITERATOR

Características

- Un objeto contenedor (agregado o colección) tal como una lista debe permitir una forma de recorrer sus elementos sin exponer su estructura interna.
- Debería permitir diferentes métodos de recorrido.
- Debería permitir recorridos concurrentes.
- No queremos añadir esa funcionalidad a la interfaz de la colección

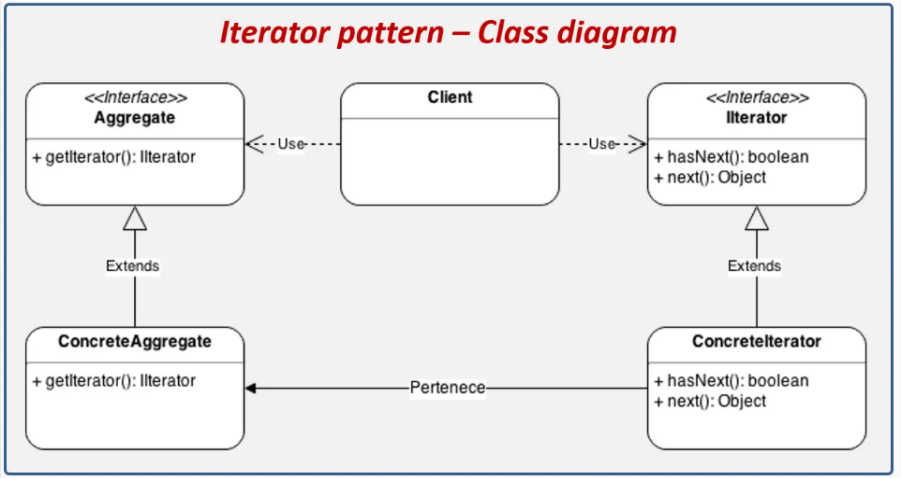


- Una clase Iterator puede definir la interfaz para acceder a una estructura de datos (Por ejemplo: una lista).

04 ITERATOR

Estructura

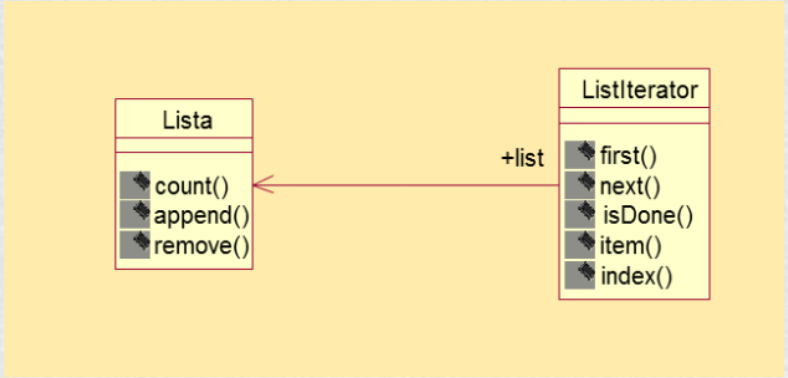
- **Client:** Actor que utiliza al Iterator.
- **Aggregate:** Interface que define la estructura de las clases que pueden ser iteradas.
- **ConcreteAggregate:** Clase que contiene la estructura de datos que deseamos iterar.
- **IIterator:** Interface que define la estructura de los iteradores, la cual define los métodos necesarios para poder realizar la iteración sobre el ConcreteAggregator.
- **ConcreteIterator:** Implementación de un iterador concreto, el cual hereda de IIterator para implementar de forma concreta cómo iterar un ConcreteAggregate.



04 ITERATOR

External Iterator

Es el recorrido controlado por el cliente, puede ser imaginado como un tipo de puntero que tiene dos operaciones primarias: referenciar un elemento particular en la colección de objetos y modificarse para apuntar al siguiente elemento. También debe existir una manera de crear un iterador para apuntar a algún primer elemento, así como alguna manera de determinar cuando el iterador ha agotado todos los elementos en el contenedor. Dependiendo del lenguaje y del uso deseado, los iteradores también pueden proporcionar operaciones adicionales o mostrar comportamientos diferentes.

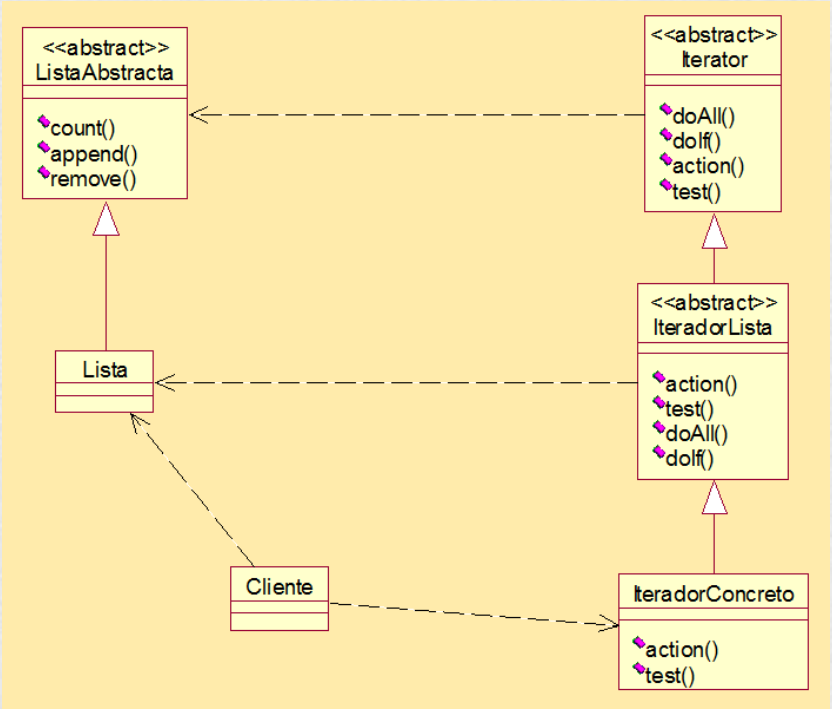


04 ITERATOR

Internal Iterator

Es el recorrido controlado por el iterador, la clase que modela la iteración ofrece métodos que controlan la ejecución: ejecutar una acción sobre todos los elementos, ejecutar una acción sobre aquellos elementos que cumplan una condición, etc.

Dados como parámetros una acción y/o condición, el método de iteración se encarga de recorrer la colección.



04 ITERATOR

Implementación

Para una correcta implementación del patrón deben considerase las siguientes cuestiones:

- ¿Quién controla la iteración? Puede realizarse de forma externa a través del cliente o realizarlo de forma interna con el Iterador.

- ¿Quién define el algoritmo de recorrido? Se puede hacer en el Iterador o en el Agregado. Si es en el caso del Agregado, se implementa el método siguiente y se deja el Iterador para almacenar la posición actual.

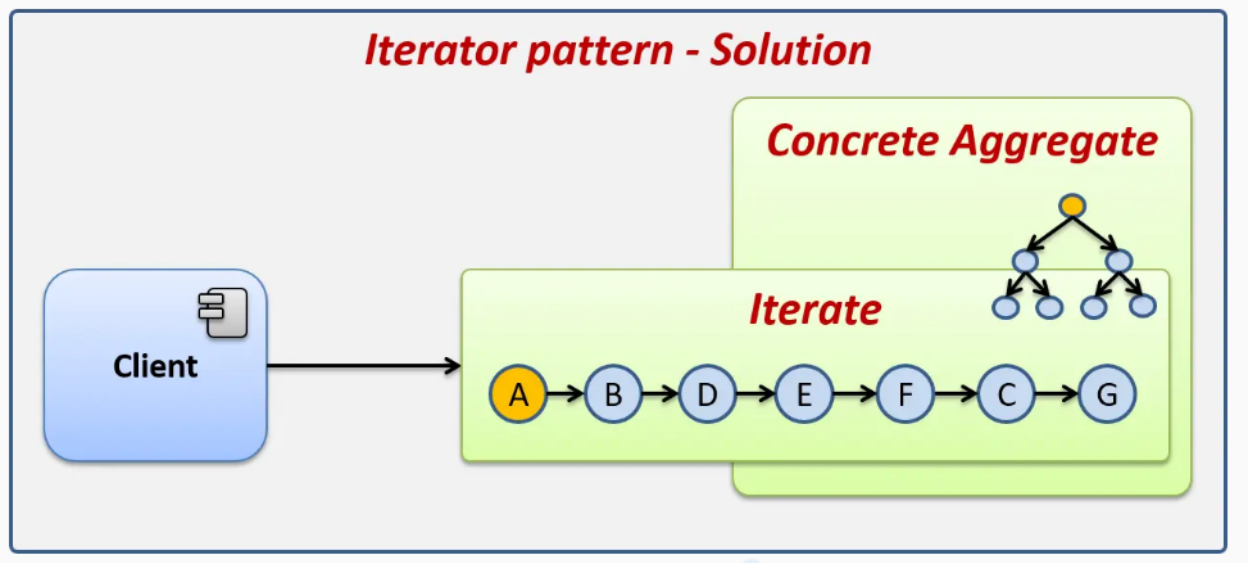
- ¿Cómo de robusto es el Iterador? Es recomendable formalizar un Iterador robusto que permita inserciones y borrados sin alterar el recorrido. Se pueden añadir operaciones adicionales al Iterador.

- Iteradores Nulos: Ayudan a manejar condiciones límite. El método Terminado() devuelve siempre True. Útil para estructuras heterogéneas.

04 ITERATOR

Ejemplo

Mediante la implementación del patrón de diseño Iterator crearemos una aplicación que nos permita recorrer una estructura organizacional jerárquica, mediante la implementación de un iterador, el cual nos permitirá recorrer todo el árbol de la estructura de forma secuencial.



¡ MUCHAS GRACIAS !