ITERATOR

D. Rdz Mtz

PATRONES DE DISEÑO

Los patrones de diseño son *soluciones encontradas* a problemas comunes en el diseño de software

Existen patrones de diseños para cualquier "nivel" en el desarrollo de software

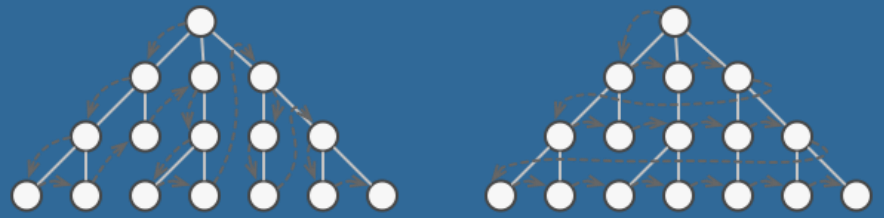
Agilizan el proceso de desarrollo, al ser estrategias conocidas y al establecer un lenguaje común entre desarrolladores

- Patrones Creacionales
- Patrones Estructurales
- Patrones de Comportamiento

ITERATOR;

Patrón de comportamiento

El Iterator proporciona una manera de acceder secuencialmente a los elementos de una colección sin necesidad de conocer su representación interna.



El Patrón Iterator propone que los detalles acerca de cómo se atraviesa una estructura de datos deben ser trasladados a un objeto "iterador" que, desde la perspectiva del *"cliente"*, simplemente produce un elemento tras otro

PROBLEMA/INTENCION

El patrón de Iterator se puede usar en situaciones en donde se tiene que iterar sobre los elementos de diferentes estructuras de datos, pero ya sea por cuestiones de legibilidad o seguridad, no se quiere revelar sus estructuras internas.

En general se puede usar cuando:

- Al "cliente" no le interesa como estan guardados los elementos, le interesa que hay elementos y se pueden recorrer.
- El cliente no necesita saber los algoritmos necesarios para recorrer la colección concreta; Pero saca los datos que necesita de diferentes tipos de contenedores

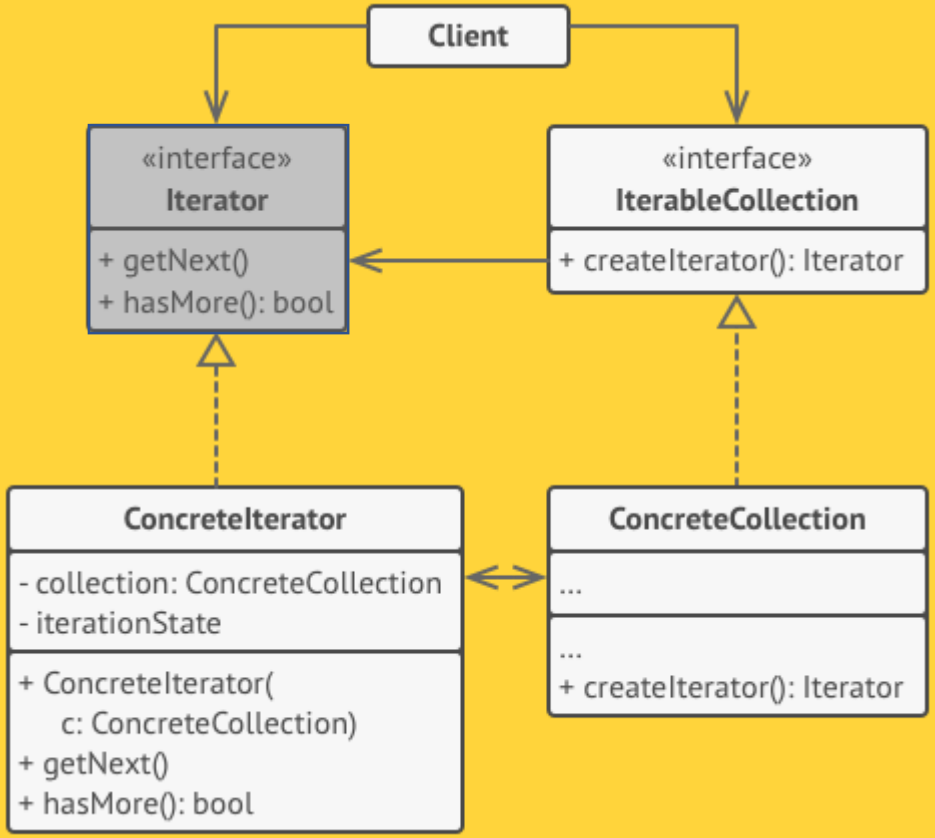
PROBLEMA/INTENCION

El Iterator encapsula los detalles concretos de la implementación, proporcionando métodos sencillos para acceder a la colección

ESTRUCTURA/MIEMBROS

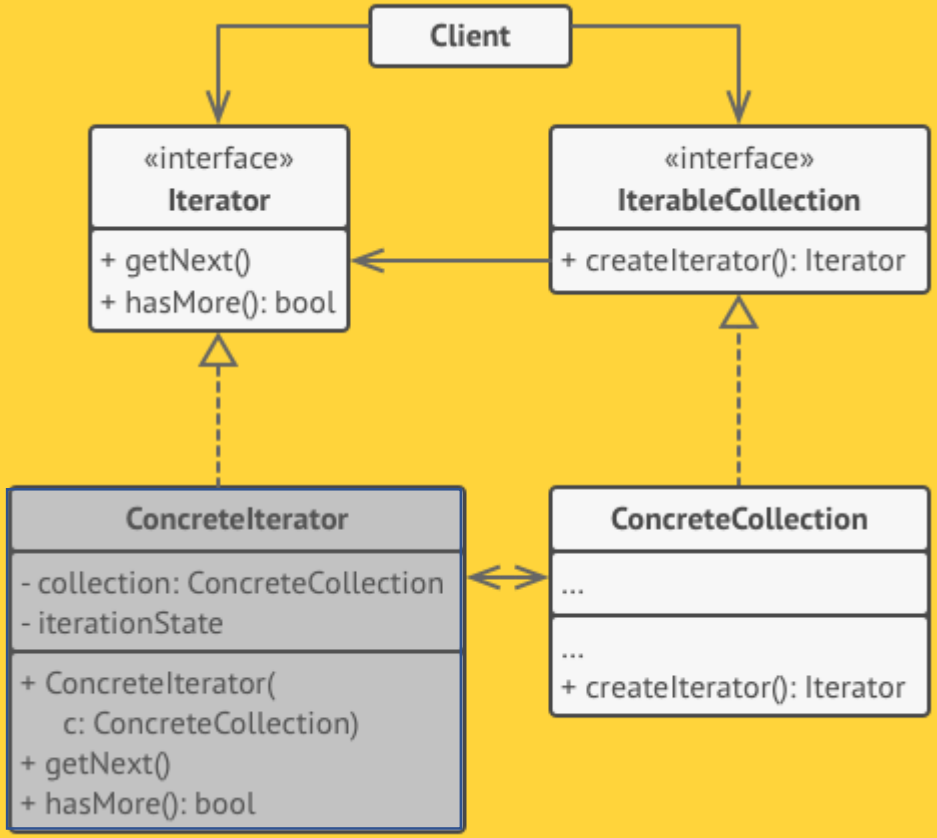
1 La interfaz **Iterator** se usa para declarar los métodos que requerimos para iterar una colección.

Como mínimo necesita un método next() y un método hasNext()



ESTRUCTURA/MIEMBROS

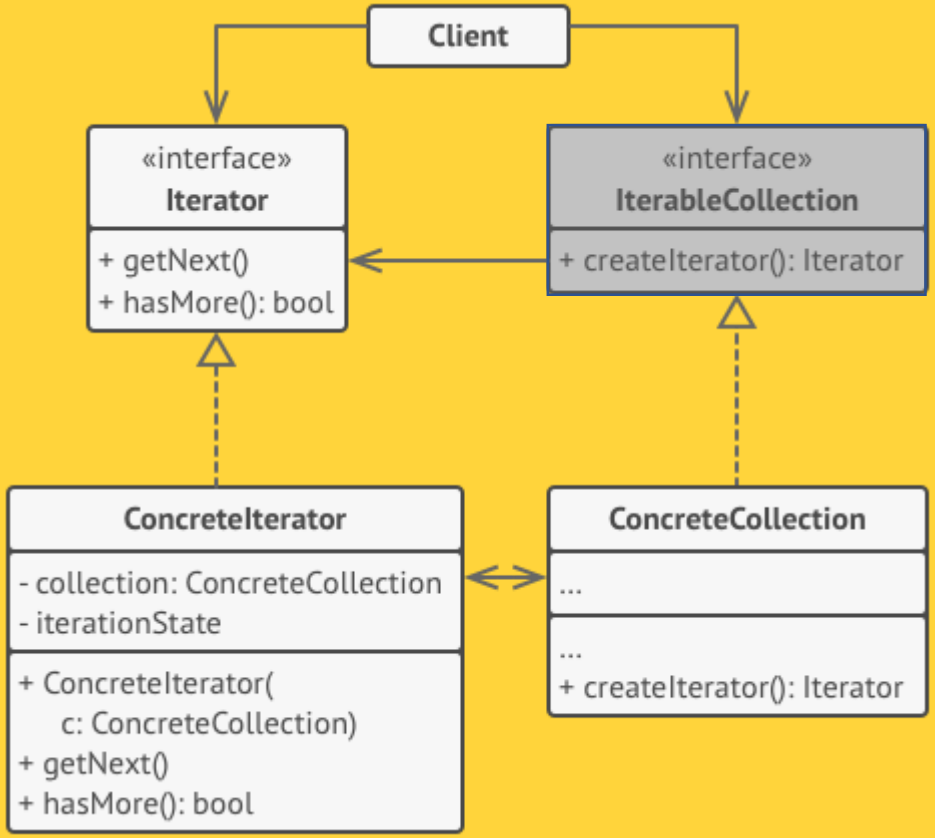
2 La clase **ConcreteIterator**, implementa Iterator, y contiene los métodos necesarios para recorrer alguna colección especifica. (Ej: un iterador para un árbol seria diferente al iterador de una lista)



ESTRUCTURA/MIEMBROS

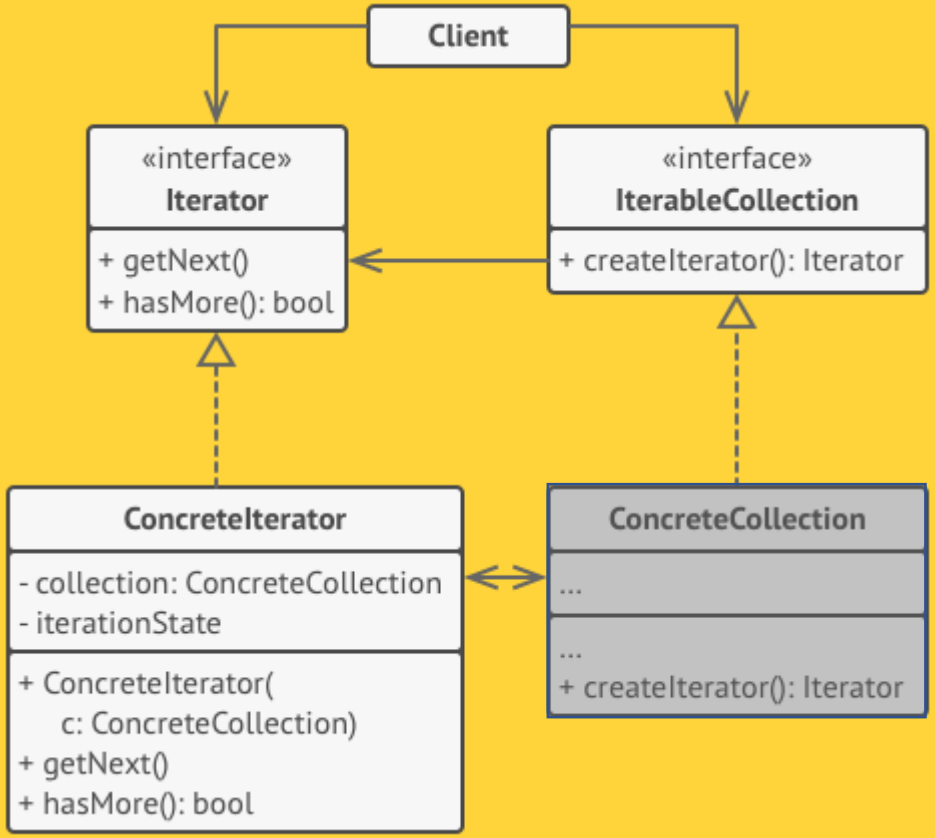
3 La interfaz **IterableCollection** declara los metodos para crear iteradores necesarios, dependiendo de cuantas formas se quiere iterar una coleccion.

Como minimo necesita un metodo createIterator()



ESTRUCTURA/MIEMBROS

4 La clase **ConcreteCollection** es el iterable concreto a usar. Retorna nuevas instancias de el ConcreteIterator cada vez que la parte del cliente mande a llamar una



CODIGO

Se adjunta el código de la presentación al commit

CODIGO

```python
# crean un iterador concreto de un iterable concreto con el metodo iter()
# (en nuestro caso seria la interfaz)

iterable = [1,2,3,4,5,6,7,8,9]
iterador = iter(iterable)

# loop infinito
while True:
    try:
        # consigue el siguiente elemento desde el iterador concreto
        i = next(iterador)

        # codigo dentro del for
        print(i)
    except StopIteration:
        # se rompe el loop cuando StopIteration sea aventado
        break
```

Aplicación o usos conocidos

Python implenta el patrón iterativo al nivel más fundamental disponible, **está incorporado en la sintaxis de Python**

Python delega el protocolo de iteración real a un par de métodos __:

- __iter__()
- __next__()