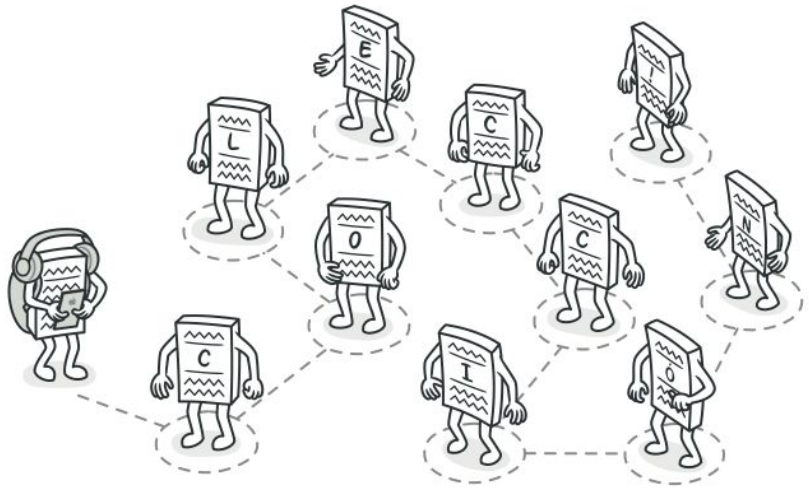
Universidad Autónoma de Chihuahua
Facultad de Ingeniería



ITERATOR

Prof. Adrián Alberto Alarcón Osollo

Johann David Gómez Villalobos 352909

Grupo 5CC2

Noviembre 09 del 2023

El patrón de diseño Iterator es un patrón de comportamiento que proporciona una forma de acceder secuencialmente a los elementos de una colección sin exponer la estructura interna de la misma. Permite a los clientes recorrer una colección de objetos sin necesidad de conocer los detalles de cómo se almacenan y organizan esos objetos.

La idea central es extraer el comportamiento de recorrido de una colección y colocarlo en un objeto independiente llamado *iterador*.

El propósito principal del patrón Iterator es proporcionar una interfaz uniforme para recorrer diferentes tipos de colecciones, ya sean listas, árboles, conjuntos u otras estructuras de datos. Al hacerlo, se logra una mayor flexibilidad y separación entre el cliente y la colección, lo que facilita el mantenimiento y la extensión del código.

Ejemplo de Uso

Supongamos que tienes una aplicación que maneja una lista de tareas pendientes. Puedes implementar el patrón Iterator para recorrer la lista de tareas sin preocuparte por cómo se almacenan internamente.

Componentes

El patrón Iterator consta de los siguientes componentes:

- Iterador (Iterator): Define una interfaz que declara métodos para recorrer elementos de la colección. Incluye métodos como next(), hasNext(), y otros que permiten al cliente acceder a los elementos de la colección de manera secuencial.

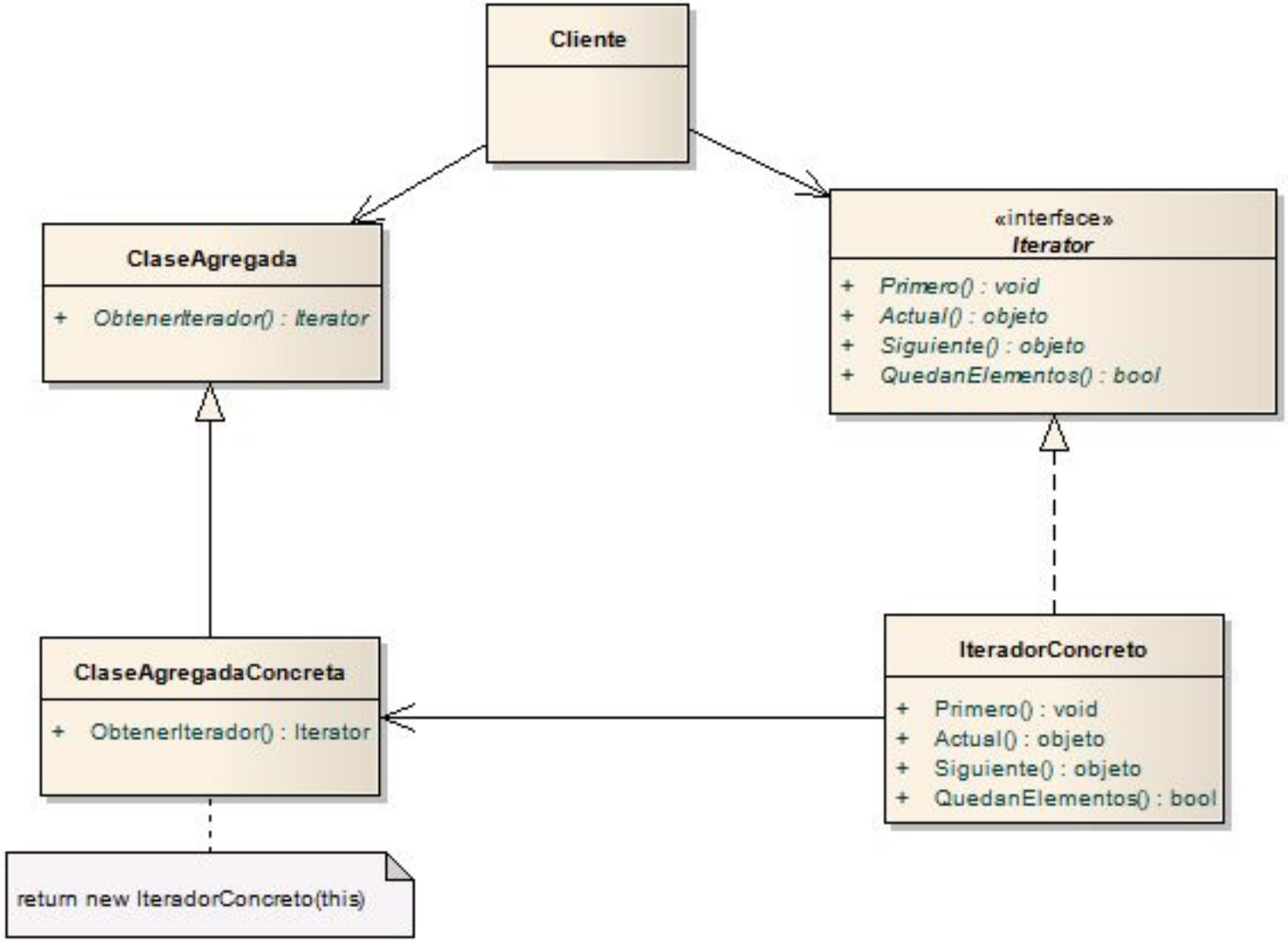
- Agregado (Aggregate): Define una interfaz o una clase abstracta que declara un método para crear un Iterador. Esta interfaz o clase puede ser implementada por diferentes tipos de colecciones, como listas, conjuntos o árboles.

- Agregado Concreto (Concrete Aggregate): Implementa la interfaz del Agregado y crea un objeto Iterador concreto. En otras palabras, proporciona una implementación específica de la colección que desea recorrer.

- Iterador Concreto (Concrete Iterator): Implementa la interfaz del Iterador y realiza el seguimiento de la posición actual en la colección. Contiene la lógica para recorrer los elementos de la colección concreta.

Ventajas	Desventajas
Separación de preocupaciones: El patrón Iterator separa el recorrido de la colección de su estructura subyacente, lo que facilita el mantenimiento y la extensión del código.	Complejidad adicional: Agregar un Iterador a una colección o estructura de datos puede aumentar la complejidad del código. Si no se implementa correctamente, puede resultar en una sobrecarga innecesaria.
Reutilización: Puedes reutilizar el mismo Iterador para diferentes colecciones, lo que ahorra tiempo y esfuerzo en la implementación.	Limitaciones de rendimiento: Dependiendo de la implementación, los iteradores pueden no ser la opción más eficiente para recorrer ciertas estructuras de datos. En algunos casos, puede ser más eficiente acceder directamente a los elementos de la colección.
Abstracción: Proporciona una interfaz uniforme para recorrer colecciones, independientemente de su tipo.	Aplicar el patrón puede resultar excesivo si tu aplicación funciona únicamente con colecciones sencillas.

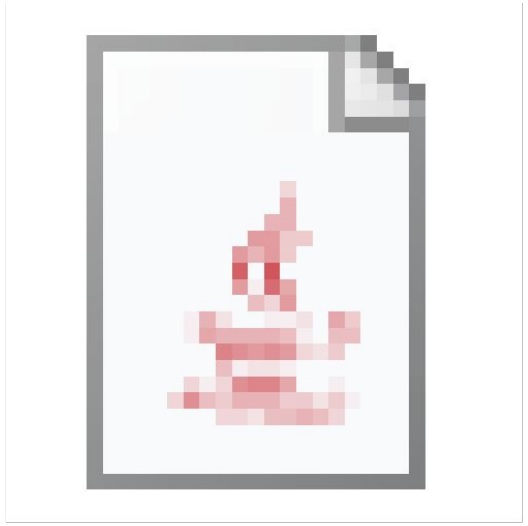
Estructura


Cliente
ClaseAgregada
+ ObtenerIterador() : Iterator
«interface»
Iterator
+ Primero() : void
+ Actual() : objeto
+ Siguiente() : objeto
+ QuedanElementos() : bool
ClaseAgregadaConcreta
+ ObtenerIterador() : Iterator
IteradorConcreto
+ Primero() : void
+ Actual() : objeto
+ Siguiente() : objeto
+ QuedanElementos() : bool
return new IteradorConcreto(this)

```java
import java.util.List;
import java.util.ArrayList;

// 1. Crear la interfaz Iterator
interface Iterador {

}

// 2. Crear la clase concreta del "Agregado"
class ListaNombres {
    private List<String> nombres = new ArrayList<>();
    private int contador = 0;

    public void llenarLista() {
        agregar("Laura");
        agregar("David");
        agregar("Kenia");
        agregar("Hector");
        agregar("Karen");
        agregar("Johann");
    }

    public void agregar(String nombre) {
        nombres.add(nombre);
        contador++;
    }

    public Iterador ObtenerIterador() {
        //Codigo para retornar un Iterador
    }

    // Clase interna que actúa como Iterador Concreto
    private class IteradorNombres implements Iterador {
        private int indice = 0;

        @Override
        public void first() {
            if (!nombres.isEmpty()) {
                System.out.println("Primer nombre de la coleccion: " + nombres.get(0) + "\n");
            }
        }

        @Override
        public Object current() {
            if (indice > 0) {
                return nombres.get(indice - 1);
```

```
        public Object current() {
            if (indice > 0) {
                return nombres.get(indice - 1);
            }
            return null;
        }

        @Override
        public Object next() {
            if (this.hasNext()) {
                return nombres.get(indice++);
            }
            return null;
        }

        @Override
        public boolean hasNext() {
            //Pista: Utilizar la variable "contador" de la clase ListaNombres
        }
    }
}

public class Iterator2 {
    public static void main(String[] args) {
        ListaNombres lista = new ListaNombres();
        lista.llenarLista();

        // Obtener un Iterador para recorrer la lista y utilizar primero() y actual()
        Iterador iterador = lista.ObtenerIterador();

        iterador.first();

        System.out.println("Comenzando Iterator \n");

        while(iterador.hasNext()) {
            Object nombre = iterador.next();
            System.out.println("Nombre siguiente: " + nombre);
            System.out.println("Recorriendo Iterator ");
            System.out.println("Nombre actual: " + iterador.actual());
            System.out.println("Procesando nombre... Listo!!!\n ");

            if(iterador.hasNext()==false){
                System.out.println("Ya recorriste toda la lista, no quedan nombres en la coleccion. ");
            }
        }
    }
}
```

```
Primer nombre de la coleccion: Laura

Comenzando Iterator

Nombre siguiente: Laura
Recorriendo Iterator
Nombre actual: Laura
Procesando nombre... Listo!!!

Nombre siguiente: David
Recorriendo Iterator
Nombre actual: David
Procesando nombre... Listo!!!

Nombre siguiente: Kenia
Recorriendo Iterator
Nombre actual: Kenia
Procesando nombre... Listo!!!

Nombre siguiente: Hector
Recorriendo Iterator
Nombre actual: Hector
Procesando nombre... Listo!!!

Nombre siguiente: Karen
Recorriendo Iterator
Nombre actual: Karen
Procesando nombre... Listo!!!

Nombre siguiente: Johann
Recorriendo Iterator
Nombre actual: Johann
Procesando nombre... Listo!!!

Ya recorriste toda la lista, no quedan nombres en la coleccion.
```



Iterator.java


GRACIAS