Taller 5
Patrones de diseño

Pablo Lara
202116655

En este documento se estudiará un proyecto de libre acceso basado en el patrón de diseño **Iterator**. El proyecto se trata de un ejemplo educativo de la implementación de este patrón de diseño. Sin embargo, existen diversas clases disponibles para ser utilizadas y accedidas al importar el proyecto como una librería. Debido a esta razón, el proyecto no posee una clase central, sino que cuenta con una amplia variedad de clases, cada una con su propia utilidad; aunque en ocasiones, se necesita la colaboración de múltiples clases para completar una tarea específica. Aquí está el enlace al repositorio: https://github.com/TheAlgorithms/Java.git .
El desafío del diseño radica en la minuciosa documentación que acompaña a cada clase, así como en la meticulosa estructuración de los datos y algoritmos en la librería. Es evidente que todo el código ha superado rigurosos estándares de calidad. Además, cada clase se encuentra distribuida en diversas carpetas, facilitando la rápida localización de información relevante y necesaria. En cuanto a la delegación de responsabilidades y colaboraciones, no percibo grandes desafíos, ya que, como mencioné previamente, las relaciones entre las clases son escasas en general, dado que están diseñadas para operar de manera independiente.

Dentro del código podemos aprecias la implementación de estructuras de datos propias del patrón iterador como lo son: CursorLinkedList, TarjansAlgorithm, LinkedQueue, Bag y DynamicArray.

El patrón iterador se emplea especialmente para encapsular información. En esencia, "proporciona un método para acceder a los elementos de un objeto agrupado secuencialmente sin revelar su estructura interna" (). La premisa fundamental de este enfoque implica trasladar la responsabilidad del acceso y recorrido de un objeto específico a otro (el iterador), evitando así exponer la organización interna del objeto que se recorre. Esta técnica posibilita múltiples recorridos sobre el mismo objeto iterable sin aumentar significativamente la complejidad del código, evitando la confusión y minimizando esfuerzos adicionales (lo que se refleja en costos económicos). Además, al aplicar un iterador de manera más abstracta, se pueden obtener los mismos beneficios para diversas estructuras (respaldando la iteración polimórfica). Por ejemplo, emplear un patrón iterador resulta especialmente valioso al realizar múltiples recorridos, como preorden e inorden, para llevar a cabo análisis complejos en árboles o grafos.

Gráficamente podemos representar con un diagrama UML la estructura base del patrón iterador:

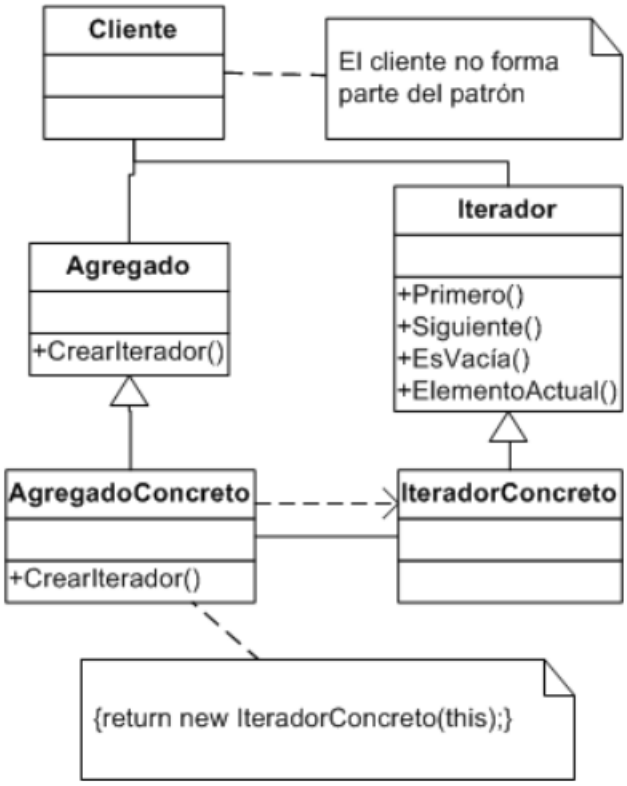

Ilustración 1 - Estructura patrón iterador

El objeto agregado tiene la capacidad de generar objetos Iterador específicos para su implementación concreta. Varias implementaciones conllevan distintos tipos de iteradores. A partir de este punto, un cliente puede interactuar con el agregado exclusivamente mediante el iterador. Si llegáramos a cambiar la forma en que se implementa el agregado (por ejemplo, si la lista se transforma en un árbol), el cliente no se vería afectado. Este patrón es altamente común y ampliamente utilizado. En el contexto de las listas, podríamos recorrerlas sin exponer su organización interna, lo que posibilitaría el uso de diferentes tipos de listas (listas estándar, basadas en vectores, etc.) y una variedad de iteradores.

En nuestro proyecto a estudiar el objeto agregado serían las diferentes estructuras de datos implementadas por el desarrollador. Teniendo esto en cuenta y lo mencionado anterior mente tiene mucho sentido haber aplicado este patrón en el marco de la librería, especialmente al implementar algoritmos como el de Tarjans, donde se necesitan realizar múltiples recorridos simultáneos en diferentes objetos mientras se registra el estado de cada uno para tomar decisiones (como encontrar los componentes fuertemente conectados en un grafo). La principal ventaja radica precisamente en esta capacidad de recorrer estructuras de manera concurrente. Otra ventaja significativa es que la clase que representa cada algoritmo iterable no está al tanto de la organización interna de los objetos recorridos (bolsas, listas, colas, entre otros). Además, es notable que emplear patrones reconocidos en la literatura y la industria es una práctica sólida que contribuye a tener un código más claro, fácil de mantener y escalable a corto, medio y largo plazo.

La principal desventaja señalada en el ReadMe del proyecto es que estas implementaciones podrían ser menos eficientes en comparación con las implementaciones nativas de Java. Por otra parte, el uso del patrón iterador podría complicar excesivamente el problema en ciertos puntos específicos, como se evidencia en la clase GrahamScan. Aquí, en realidad, no se requiere utilizar la estructura iterable Stack, ya que no se realizan recorridos múltiples en ningún momento. Aunque no se considera un error, perfeccionar el código hasta este nivel (siguiendo las mejores prácticas en todo momento) podría resultar costoso en circunstancias particulares.

Si se optara por no emplear el patrón, implicaría modificar el código en ciertas secciones. En el algoritmo de Tarjans, posiblemente se habría abordado el problema copiando el objeto a recorrer en múltiples variables para cada recorrido simultáneo necesario, o se habrían creado

ciclos anidados, lo que habría incrementado la complejidad del algoritmo. Esto podría resultar no solo subóptimo, sino incluso ineficaz en situaciones extremas. Se podrían notar las desventajas mencionadas anteriormente en estos enfoques alternativos.