Nombre: Carlos Fernando Díaz Vargas

Código: 202210262

Taller 5 (Patrón iterador)

- El proyecto que escogí hace parte de un gran número de proyectos publicados por TheAlgoritms, que es un perfil enfocado a publicar la implementación de los algoritmos y estructuras de datos más relevantes hasta el momento en diferentes lenguajes de programación (en este caso Java). Las implementaciones son de código abierto y en su mayoría tienen propósitos educativos, por lo que, puede que tengan un rendimiento ligeramente peor a las librerías estándar de Java. Sin embargo, hay muchas clases muy interesantes que pueden ser usadas y accedidas si se importa el proyecto como una librería. Por esto mismo, el proyecto no tiene una clase core, sino que tiene multiplicidad de clases con distinta utilidad cada una, aunque en muchas ocasiones, se requiere de la colaboración de varias clases para llevar a cabo una tarea específica. El link del repositorio es el siguiente: https://github.com/TheAlgorithms/Java.git.

  Por otra parte, considero que el mayor reto del diseño es la documentación tan extensa que tiene cada una de las clases y lo bien pensada que está la estructura de cada una de las estructuras de datos y algoritmos de la librería. Se nota que todo el código ha pasado por filtros de calidad. Asimismo, cada clase se encuentra de múltiples carpetas, lo que ayuda a localizar la información útil y necesaria de manera efectiva y, sobre todo rápida. No creo que haya habido grandes retos a nivel de delegación de responsabilidades y colaboraciones, porque como ya mencioné, en

general, hay pocas relaciones entre las clases, precisamente porque están pensadas para funcionar de manera independiente.

- Me enfoqué en la implementación de algunas estructuras de datos y algoritmos que usan precisamente el patrón iterador. En las que me concentré para realizar este texto son: Bag, LinkedQueue, DynamicArray, CursorLinkedList y TarjansAlgorithm

En todas las clases mencionadas, el iterador que se toma como base se encuentra en la librería nativa de Java, utils. La interfaz Iterator de esta librería tiene como base 3 métodos relevantes para este patrón:

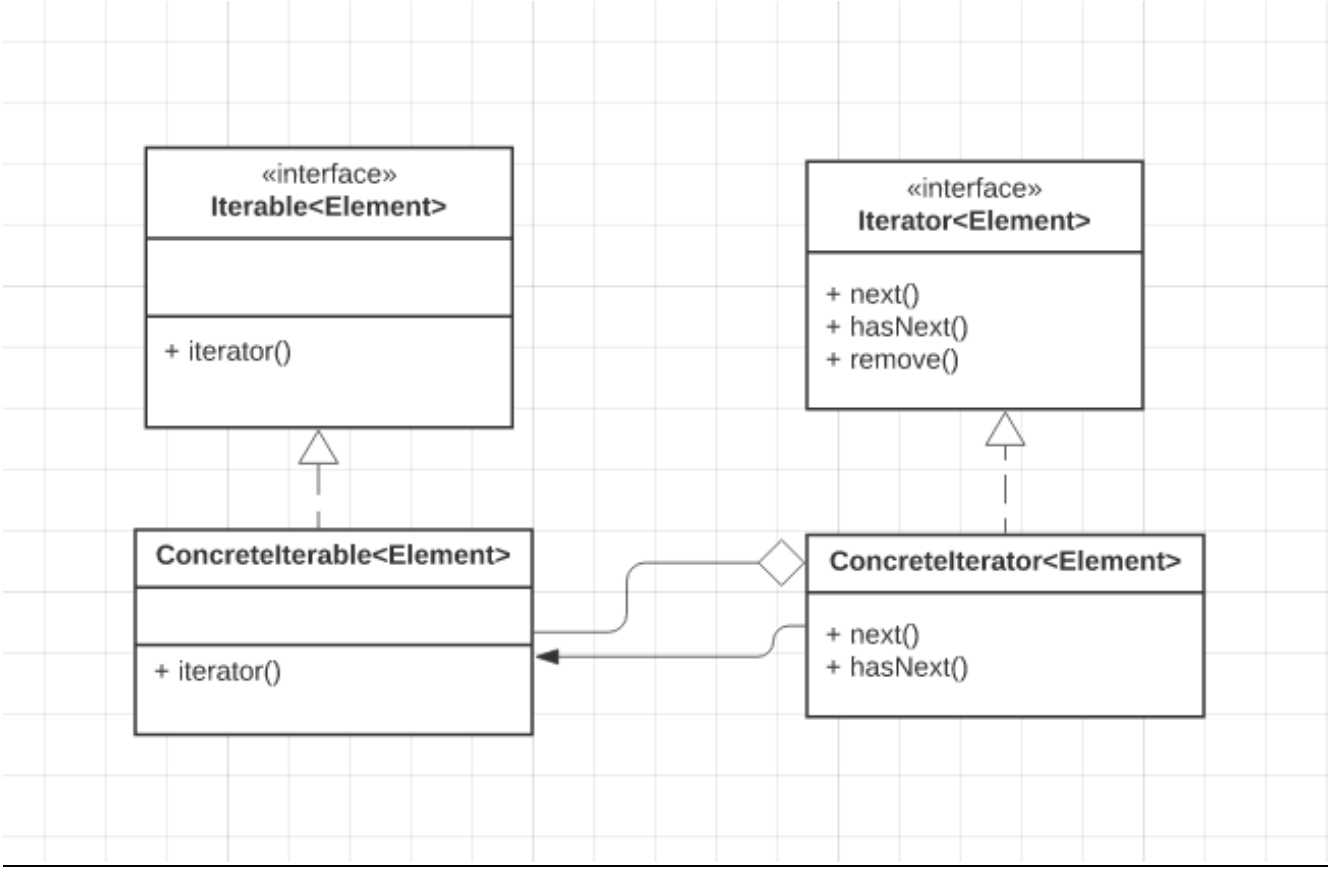
1. hasNext()
2. next()
3. remove()

Estos métodos se encargan de retornar verdadero si aún quedan elementos por iterar, retornar el valor del siguiente elemento en la iteración y eliminar el último elemento retornado por el iterador respectivamente. Por otro lado, las clases que son las que se van a iterar, deben implementar la interfaz Iterable de la misma librería de java, con la que se evidencia la estructura más común del patrón mostrada más adelante.

- El patrón iterador se usa especialmente como una manera de encapsular información. En resumidas cuentas, “el patrón iterador proporciona una forma de acceder a los elementos de un objeto agregado secuencialmente sin exponer su representación subyacente” () La idea principal de este patrón es delegar la responsabilidad del acceso y recorrido de un objeto particular a otro (el objeto iterador), para no exponer la estructura interna del objeto recorrido, además de permitir realizar varios recorridos al

tiempo de distintas formas sobre el mismo objeto iterable, sin tener que usar muchas más líneas de código, evitando desorden y esfuerzo (que se traduce en costes económicos) innecesarios. Además de todo lo anterior, usar un iterador a un nivel más abstracto, también permite obtener los beneficios anteriormente mencionados para múltiples estructuras (soportar iteración polimórfica). Por ejemplo, usar un patrón iterador puede ser particularmente útil cuando se requieren hacer varios recorridos como preorden e inorden para realizar análisis complejos sobre árboles o grafos.

La estructura básica de un patrón iterador se ve como la siguiente:



Se observa que el iterador concreto y el aggregate mantienen una relación de delegación, es decir, hay acoplamiento. Podría parecer anti intuitivo, pero otro objetivo fundamental de este patrón es reducir el acoplamiento, al evitar la necesidad de crear múltiples interfaces para recorrer distintas estructuras o crear métodos en distintas clases para realizar los recorridos.

En el diagrama anterior, “Iterator” es la interfaz encargada de acceder y recorrer los elementos, “ConcreteIterator” implementa la interfaz “Iterator” y es necesaria para recorrer una estructura de datos en particular, “Aggregate” es la interfaz que contiene un método para crear un objeto de la clase “Iterator” y “ConcreteAggregate” es la interfaz que debe implementar la estructura que se va a recorrer para que tenga un iterador de base asociado, aunque puede tener más de uno.

En algunas implementaciones el control de la iteración la tiene el propio cliente (el que usa el iterador), mientras que en otras es el mismo iterador el que tiene el control. En el primer caso, el iterador se llama externo, mientras que en el segundo, se denomina iterador interno.

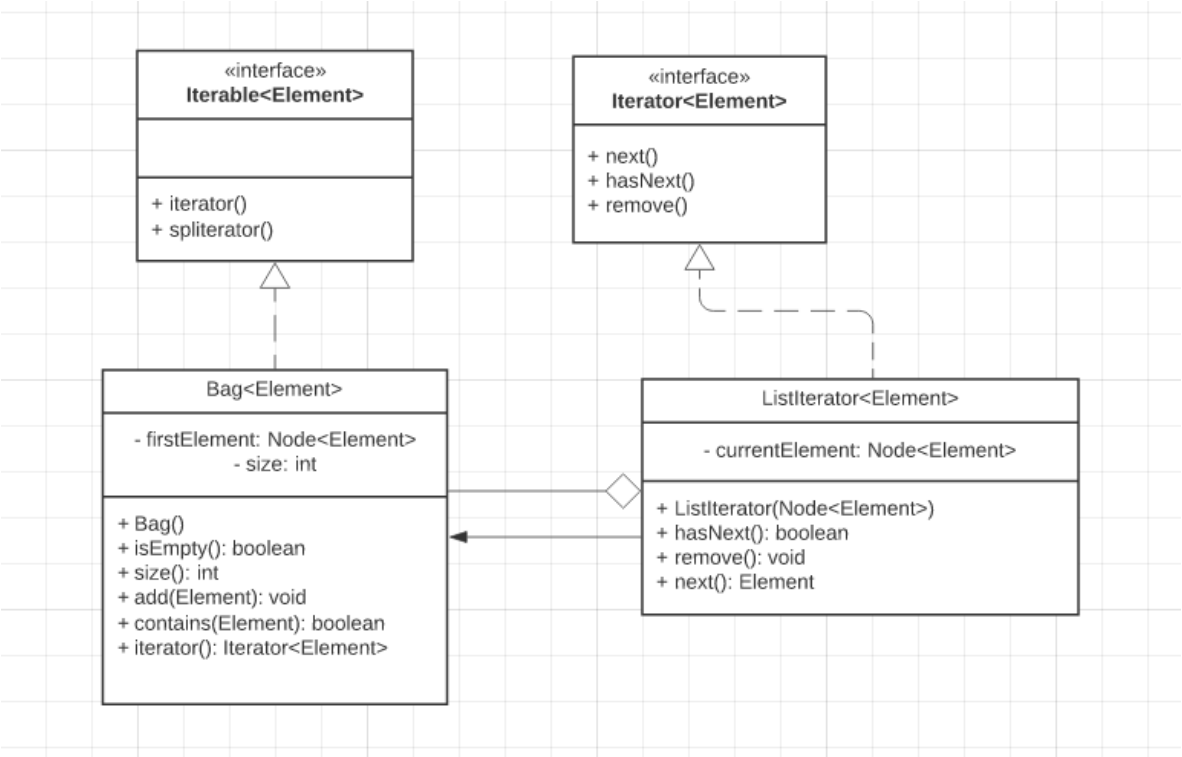
Asimismo, cuando se crea un iterador robusto, las modificaciones que se realicen sobre el cliente que usa el iterador no son tenidas en cuenta al momento de realizar una iteración en tiempo real, evitando errores como leer, eliminar o modificar valores indeseados por cambios en la estructura del iterable.

- Tiene mucho sentido haber usado el patrón en el contexto de la librería, porque en la implementación de algoritmos como el de Tarjans, se requieren hacer múltiples recorridos sobre distintos objetos al tiempo, mientras se mantiene una traza del estado de cada uno, para poder tomar decisiones (en este caso, encontrar

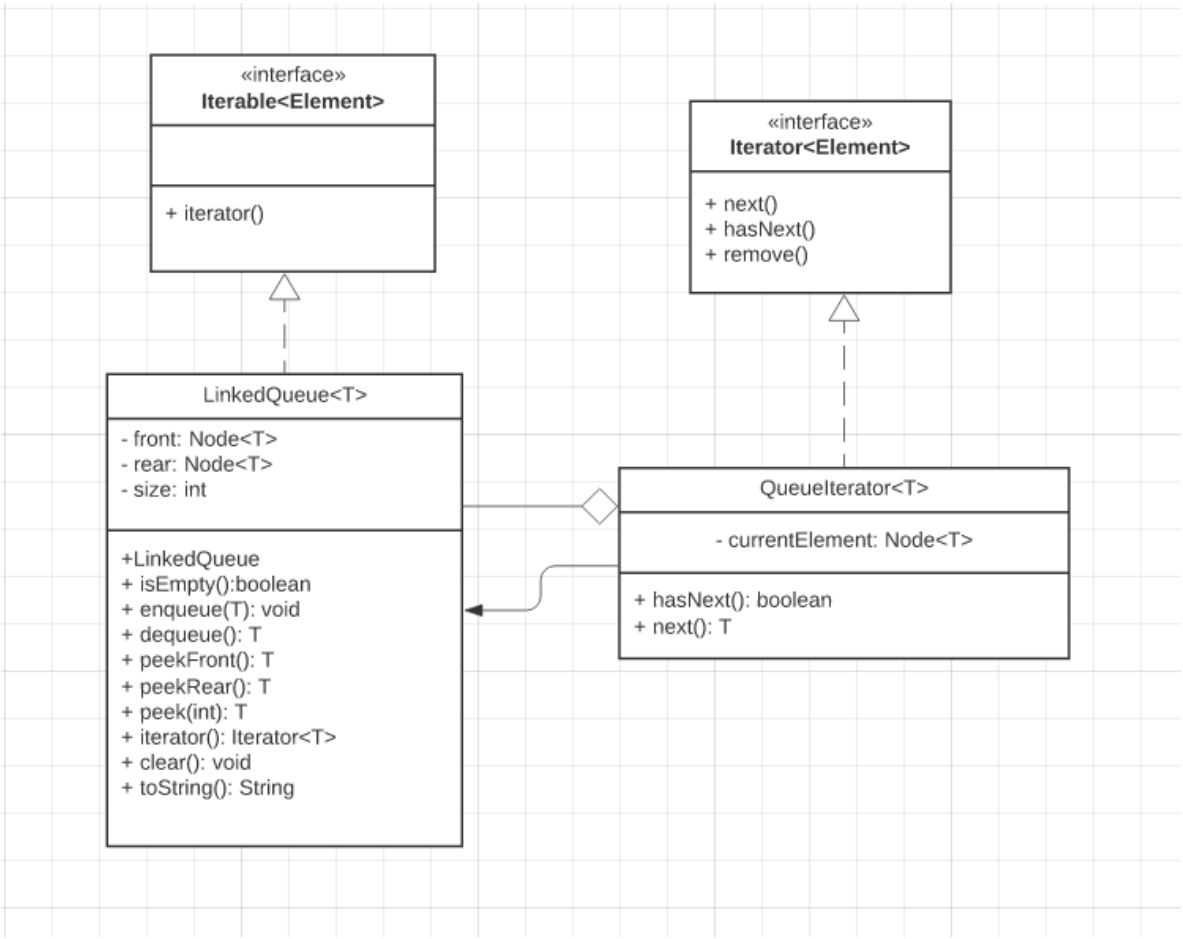
los componentes fuertemente conectados en un grafo). La principal ventaja es esta misma, poder recorrer estructuras de distintas maneras al tiempo. Otra ventaja importante es que la clase que representa cada uno de los algoritmos que tiene iterables, no conoce la representación interna de los objetos recorridos (bags, listas, queues, entre otros). Asimismo, es destacable que el uso de patrones reconocidos en la literatura y en la industria es una buena práctica y ayuda a tener un código más legible, mantenible y extensible a corto, mediano y largo plazo.

Entrando en detalle, es vital mencionar que la creación de ambas interfaces (Iterable e Iterator) depende del tipo de estructura que se desea recorrer. Por defecto, estas interfaces tienen iteradores para listas, valores dobles, enteros y longs.

La relación para las clases de Bag, y LinkedQueue mencionadas anteriormente es la siguiente:



Como se puede ver, en este caso el “ConcreteIterable” es el objeto de clase Bag y el “ConcreteIterator” asociado es ListIterator, que se crea en el mismo archivo .java que el Bag.



La implementación de LinkedQueue no es muy diferente a la de Bag. La mayor diferencia es que ahora se necesitan sobreescribir solo hasNext() y next() para que le estructura de datos pueda ser recorrida correctamente.

- La principal desventaja que se advierte en el ReadMe del proyecto es que estas implementaciones pueden resultar más ineficientes comparadas con las nativas de Java. Por otro lado, usar el patrón iterador puede sobre complejizar el problema en algunos puntos particulares como por ejemplo en la clase GrahamScan, donde en realidad no es necesario usar la estructura iterable Stack, porque en ningún momento se hacen recorridos múltiples. Aunque no considero que sea un error,

puede que pulir el código a este punto (buenas prácticas en todo momento) puede resultar costoso en circunstancias particulares.

```
    public Iterable<Point> hull() {
        Stack<Point> s = new Stack<>();
        for (Point p : hull) s.push(p);
        return s;
    }
```

- Se pudo haber decidido no adoptar el patrón, lo cual implicaría cambiar el código en algunas partes. En el algoritmo de Tarjans probablemente se hubiera solucionado el problema copiando el objeto a recorrer en tantas variables como distintos recorridos se necesitara hacer al tiempo o, también se podía crear ciclos anidados haciendo que la complejidad del algoritmo aumente y, posiblemente no solo no sea óptimo, sino volviéndose inútil en algunos de los casos más extremos. Se pueden notar las desventajas descritas anteriormente.