“INSTITUTO POLITÉCNICO NACIONAL”

(Unidad Profesional Interdisciplinaria de Ingeniería Zacatecas)

Investigación ArrayList - LinkedList

Programación Orientada a Objetos

Alumno: • Daniel Torres Montañez

Grupo: 2CM1

Docente: Roberto Oswaldo Cruz Leija

24 de octubre del 2019

INTERFACE LIST

La interface List la hemos venido utilizando aunque quizás sin ser plenamente conscientes de ello. Esta interface es la encargada de agrupar una colección de elementos en forma de lista, es decir, uno detrás de otro. En una lista los elementos pueden ser accedidos por un índice que indica la posición del elemento en la colección.

LIST

Esta interfaz también conocida como “secuencia” normalmente acepta elementos repetidos o duplicados, y al igual que los arrays es lo que se llama “basada en 0”. Esto quiere decir que el primer elemento no es el que está en la posición “1”, sino en la posición “0”.

Esta interfaz proporciona debido a su uso un iterador especial (la interfaz Iterator e Iterable las hemos podido conocer en anteriores entregas) llamada ListIterator. Este iterador permite además de los métodos definidos por cualquier iterador (recordemos que estos métodos son hasNext, next y remove) métodos para inserción de elementos y reemplazo, acceso bidireccional para recorrer la lista y un método proporcionado para obtener un iterador empezando en una posición específica de la lista.

Debido a la gran variedad y tipo de listas que puede haber con distintas características como permitir que contengan o no elementos null, o que tengan restricciones en los tipos de sus elementos, hay una gran cantidad de clases que implementan esta interfaz.

A modo de resumen vamos a mencionar las clases más utilizadas de acuerdo con nuestra experiencia.

- ArrayList.

- LinkedList.

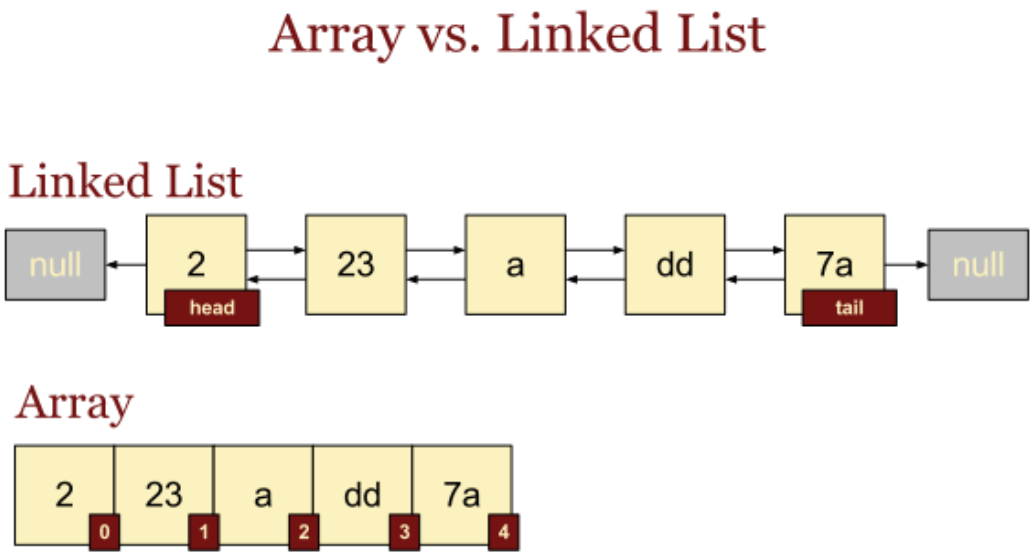
- Stack.

- Vector.

LinkedList y ArrayList

son dos diferentes implementaciones de la interfaz List. LinkedList usa internamente una lista doblemente ligada, mientras que ArrayList usa un arreglo que cambia de tamaño dinámicamente.

Podemos visualizarlas de la siguiente forma:



LinkedList permite **eliminar e insertar elementos en tiempo constante** usando iteradores, pero el acceso es secuencial por lo que encontrar un elemento toma un tiempo proporcional al tamaño de la lista. Normalmente la complejidad de esa operación promedio sería O(n/2) sin embargo usar una lista doblemente ligada el recorrido puede ocurrir desde el principio o el final de la lista por lo tanto resulta en O(n/4).
Por otro lado ArrayList ofrece **acceso en tiempo constante** O(1), pero si quieres añadir o remover un elemento en cualquier posición que no sea la última es necesario mover elementos. Además si el arreglo ya está lleno es necesario crear uno nuevo con mayor capacidad y copiar los elementos existentes.

LA CLASE ARRAYLIST

Vamos a hablar brevemente sobre todas las clases anteriores, pero vamos a comenzar por ArrayList que ha sido una de las clases en las que hemos venido trabajando más a menudo por lo que ya conocemos parte de ella.

ArrayList como su nombre indica basa la implementación de la lista en un array. Eso sí, un array dinámico en tamaño (es decir, de tamaño variable), pudiendo agrandarse el número de elementos o disminuirse. Implementa todos los métodos de la interfaz List y permite incluir elementos null.

Un beneficio de usar esta implementación de List es que las operaciones de acceso a elementos, capacidad y saber si es vacía o no se realizan de forma eficiente y rápida. Todo arraylist tiene una propiedad de capacidad, aunque cuando se añade un elemento esta capacidad puede incrementarse. Java amplía automáticamente la capacidad de un arraylist a medida que va resultando necesario.

A través del código podemos incrementar la capacidad del arraylist antes de que este llegue a llenarse usando el método ensureCapacity. Esta clase no es sincronizada lo que entre otras cosas significa que si hay varios procesos concurrentes (procesos que se ejecutan al mismo tiempo) sobre un objeto de este tipo y en dos de ellos se modifica la estructura del objeto se pueden producir errores.

LINKEDLIST

La clase LinkedList implementa la interface List. Eso quiere decir que tendrá una serie de métodos propios de esta interface y comunes a todas las implementaciones. Así utilizando siempre que se pueda declaración de objetos del tipo definido por la interface podemos cambiar de forma relativamente fácil su implementación (por ejemplo pasar de ArrayList a LinkedList y viceversa) y conseguir mejoras en el rendimiento de nuestros programas con poco esfuerzo.

Ahora centrándonos en la clase LinkedList, ésta se basa en la implementación de listas doblemente enlazadas. Esto quiere decir que la estructura es un poco más compleja que la implementación con ArrayList, pero… ¿Qué beneficios nos aporta si la estructura es más compleja?.

¿Rapidez?, pues no mucha la verdad, de hecho ArrayList es la favorita para realizar búsquedas en una lista y podríamos decir que ArrayList es más rápida para búsquedas que LinkedList. Entonces, ¿qué interés tiene LinkedList?. Si tenemos una lista y lo que nos importa no es buscar la información lo más rápido posible, sino que la inserción o eliminación se hagan lo más rápidamente posible, LinkedList resulta una implementación muy eficiente y aquí radica uno de los motivos por los que es interesante y por los que esta clase se usa en la programación en Java.

La clase Linkedlist no permite posicionarse de manera absoluta (acceder directamente a un elemento de la lista) y por tanto no es conveniente para búsquedas pero en cambio sí permite una rápida inserción al inicio/final de la lista y funciona mucho más rápido que Arraylist por ejemplo para la posición 0 (imaginemos que en un ArrayList deseamos insertar en la posición 0 y tenemos muchos elementos, no solo tendremos que realizar la operación de insertar este nuevo elemento en la posición 0 sino que tendremos que desplazar el elemento que teníamos de la posición 0 a la 1, el que estaba anteriormente en la posición 1 a la 2 y así sucesivamente… Si tenemos un gran número de elementos la operación de inserción es más lenta que en la implementación LinkedList, donde el elemento simplemente “se enlaza” al principio, sin necesidad de realizar desplazamientos en cadena de todos los demás elementos).

LinkedList

Operación	Complejidad Promedio
`get`	O(n/4)
`add(E element)`	**O(1)**
`add(int index, E element)`	O(n/4)
`remove(int index)`	O(n/4)
`Iterator.remove()`	**O(1)**
`ListIterator.add(E element)`	**O(1)**

ArrayList

Operación	Complejidad Promedio
`get`	**O(1)**
`add(E element)`	**O(1)**
`add(int index, E element)`	O(n/2)
`remove(int index)`	O(n/2)
`Iterator.remove()`	O(n/2)
`remove(int index)`	O(n/2)

Estas métodos son aplicables a las clases genéricas de Java LinkedList y ArrayList sin embargo estos tipos no son exclusivos de Java, es posible encontrar implementaciones en otros lenguajes de programación, por ejemplo en C# existen LinkedList y List.

Ventajas y desventajas

LinkedList

Ventajas	Desventajas
Añadir y remover elementos con un iterador	Uso de memoria adicional por las referencias a los elementos anterior y siguiente
Añadir y remover elementos al final de la lista	El acceso a los elementos depende del tamaño de la lista

ArrayList

Ventajas	Desventajas
Añadir elementos	Costos adicionales al añadir o remover elementos
Acceso a elementos	La cantidad de memoria considera la capacidad definida para el ArrayList, aunque no contenga elementos

Referencias

http://www.enrique7mc.com/2016/07/diferencia-entre-arraylist-y-linkedlist/

https://www.w3schools.com/java/java_arraylist.asp