Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria de Ingeniería
campus Zacatecas

Ingeniería en Sistemas Computacionales

Programación orientada a objetos

Grupo: 2CM1

Partes de un sistema operativo

Christian Iván Aguilar Valdez

Roberto Oswaldo Cruz Leija

Zacatecas, Zac., a 24 de octubre de 2019

Definición de List, ArrayList y LinkedList

- 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.
  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.
- ArrayList:
  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:
  Para una clase LinkedList tendremos una lista enlazada en la que los elementos se añaden en cualquier parte de la lista muy fácilmente. Aquí, para encontrar un elemento hay que recorrer la lista. Es eficiente cuando el tamaño fluctúa y sobre todo en posiciones centrales.

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).

Diferencias

List es una colección, y una colección puede ser interfaces y clase abstracta que nos permite identificar los objetos independientemente de la implementación. Es decir, son genéricas.

En base a esto, List es una interfaz genérica que representa una colección ordenada de elementos que pueden repetirse.

Mientras, dos listas de propósito general serían las clases LinkedList y ArrayList y de propósito específico, Vector y CopyOnWriteArrayList.

Así pues, lo mas conveniente sería comparar LinkedList y ArrayList.

LinkedList	**ArrayList**
Permite añadir y remover elementos con un iterador	La cantidad de memoria considera la capacidad definida para el ArrayList, aunque no contenga elementos
Permite añadir y remover elementos al final de la lista	Costos adicionales al añadir o remover elementos
Tamaño ilimitado	Cuando se supera el tamaño del array, se crea uno nuevo más grande y se copian en él los elementos del antiguo
Uso de memoria adicional por las referencias a los elementos anterior y siguiente	Si especifica el tamaño de un ArrayList al momento de crearlo se disminuye la cantidad de memoria usada porque no necesita aumentar de tamaño y copiar los elementos
El acceso a los elementos depende del tamaño de la lista	Facilidad al añadir y acceder a elementos

En la práctica la mayoría de las ocasiones es mejor usar ArrayList porque el tiempo de las operaciones y uso de memoria es menor que en LinkedList, de manera simple: si no sabes cual usar usa ArrayList.

Eso no quiere decir que LinkedList nunca se utilice, existen algunos casos muy específicos donde es la mejor opción, por ejemplo la pila de llamadas del lenguaje C está implementada usando una estructura de datos con estas características.

Bibliografía

- Diferencia entre ArrayList y LinkedList. (2019). Retrieved 24 October 2019, from http://www.enrique7mc.com/2016/07/diferencia-entre-arraylist-y-linkedlist/
- Curso de Java. Estructuras de datos: ArrayList y LinkedList. (2019). Retrieved 24 October 2019, from https://www.redeszone.net/2012/03/05/curso-de-java-estructuras-de-datos-arraylist-y-linkedlist/
- Alejandro, C., & Giorgi, A. (2019). ¿List o ArrayList en Java?. Retrieved 24 October 2019, from https://es.stackoverflow.com/questions/171467/list-o-arraylist-en-java/171472