INTERFACE LIST. CLASE LINKEDLIST DEL API JAVA. EJERCICIO RESUELTO Y DIFERENCIAS ENTRE ARRAYLIST Y LINKEDLIST

INTERFACE LIST

Vamos a continuar el estudio de la interface List del api de Java, pero esta vez usaremos la clase LinkedList como forma de instanciar la interface. También veremos las características más importantes de LinkedList, las diferencias que tiene con ArrayList y haremos un ejemplo a modo de ejercicio.

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).

NO SÉ QUE IMPLEMENTACION UTILIZAR ¿ARRAYLIST O LINKEDLIST?

Es posible que en un programa tengamos que usar una lista pero no sepamos si es más conveniente usar ArrayList ó LinkedList. En este caso podemos hacer lo siguiente:

Si a priori pensamos que es mejor utilizar una implementación con ArrayList porque pensamos que las búsquedas van a ser la mayoría de las operaciones, entonces pondremos algo así:

List listaObjetos = new ArrayList();

Por el contrario si pensamos a priori que la mayoría de las operaciones sobre esta lista de objetos van a ser inserciones o eliminaciones sobre listas grandes escribiremos:

List listaObjetos = new LinkedList();

Aquí queda reflejada la utilidad que tiene el uso de interfaces, porque usando los mismos métodos, podemos tener varias implementaciones que nos permitirán un mejor rendimiento dependiendo del uso que demos a nuestra aplicación.

En caso de no tener claro qué operación va a ser más frecuente en general usaremos ArrayList.

EJERCICIO COMPARATIVO ARRAYLIST VS LINKEDLIST

Vamos a realizar a continuación un ejercicio comparativo, donde vamos a observar y medir el tiempo que tarda en realizarse una inserción en una lista ArrayList y en otra lista LinkedList.

Para ello y basándonos en la clase Persona que hemos venido utilizando a lo largo del curso, escribiremos este código en nuestro editor:

```
/* Ejemplo Interface List aprenderaprogramar.com */

public class Persona{

   private int idPersona;
   private String nombre;
   private int altura;

   public Persona(int idPersona, String nombre, int altura) {
      this.idPersona = idPersona;
      this.nombre = nombre;
      this.altura = altura;}

   @Override
   public String toString() {
      return "Persona-> ID: "+idPersona+" Nombre: "+nombre+" Altura: "+altura+"\n";
   }

}
```

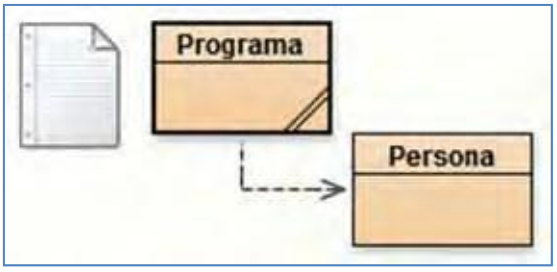
Y nuestra clase con el método main:

```
/* Ejemplo Interface List aprenderaprogramar.com */

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class Programa {
   public static void main(String arg[]) {
        List<Persona> listaarray = new ArrayList<Persona>();
        List<Persona> listalinked = new LinkedList<Persona>();
        long antes;
        for(int i=0;i<10000;i++)
        {
           listaarray.add(new Persona(i,"Persona"+i,i)); // En este ejemplo cada persona lleva datos ficticios
           listalinked.add(new Persona(i,"Persona"+i,i));
        }
        System.out.println("Tiempo invertido en insertar una persona en listaarray (en nanosegundos):");
        antes = System.nanoTime();
        listaarray.add(0,new Persona(10001,"Prueba",10001)); // Inserción en posicion 0 de una persona
        System.out.println(System.nanoTime()- antes);
        System.out.println("Tiempo invertido en insertar una persona en listalinked (en nanosegundos):");
        antes = System.nanoTime();
        listalinked.add(0,new Persona(10001,"Prueba",10001)); // Inserción en posicion 0 de una persona
        System.out.println(System.nanoTime()- antes);
    }
}
```

Nuestro diagrama de clases es el siguiente:



Como podemos observar en el código de la clase Programa, creamos 2 listas, una basada en la implementación ArrayList y otra basada en LinkedList. A continuación, declaramos una variable de tipo long para contar el tiempo invertido en cada inserción.

Previamente a esto, cargamos en las listas respectivamente 10.000 personas lo cual no es ni mucho menos una gran cantidad (piensa por ejemplo en una empresa de telefonía móvil, puede tener muchos miles de clientes), pero tampoco queremos alargar mucho el proceso de carga en este ejemplo.

Bien, pues una vez cargadas nuestras listas: listaarray y listalinked, medimos el periodo de tiempo que consume una inserción en la posición 0 en cada implementación y obtenemos los siguientes resultados.

Obtenemos la siguiente salida por consola (esta salida será diferente en cada ejecución, en lo que queremos fijarnos es en que existe una diferencia):

```
Tiempo invertido en insertar una persona en listaarray (en nanosegundos):
5400
Tiempo invertido en insertar una persona en listalinked (en nanosegundos):
4000
------------------------------------------------------------------------
BUILD SUCCESS
```

Como vemos el tiempo invertido en insertar una persona en la posición inicial en una lista implementada con ArrayList es mayor que el empleado en una LinkedList. Si este proceso tuviéramos que hacerlo repetitivamente, el coste en tiempo podría ser significativamente diferente y motivo suficiente para usar una implementación u otra.

Ahora cambiaremos el código de programa, para insertar a la nueva persona en la posición 5000 en vez de en la posición 0, lo que hacemos con este código (*listaarray.add(5000,new Persona(10001,"Prueba",10001));* y *listalinked.add(5000,new Persona(10001,"Prueba",10001));*) y tras ejecutar de nuevo el programa obtendremos una pantalla algo similar a la siguiente:

```
Tiempo invertido en insertar una persona en listaarray (en nanosegundos):
6100
Tiempo invertido en insertar una persona en listalinked (en nanosegundos):
42000
------------------------------------------------------------------------
BUILD SUCCESS
```

Vemos que en esta ocasión el tiempo en insertar con listaarray es más o menos el mismo que en el anterior ejemplo, pero que con listalinked insertar en la posición media de la lista es tremendamente más lento. Esto es debido a que para insertar en esa posición la implementación con linkedlist debe recorrer parte de la lista hasta llegar a esa posición, mientras que con arraylist tenemos acceso directo a una posición determinada. Todo esto está relacionado con la forma en que está implementado el código del api de Java, aspectos que no tenemos por qué conocer pero que afectan al rendimiento o idoneidad de una clase u otra. Aunque no tengamos que conocer el código del api Java, sí debemos ser conscientes de las ventajas e inconvenientes que puede tener el uso de una u otra clase como programadores.

Ahora para finalizar vamos a utilizar el método add sin una posición determinada, pero que por defecto añade el nuevo elemento en la última posición.

Por tanto volvemos a modificar el código de la clase Programa y sustituiremos las líneas *listaarray.add(5000,new Persona(10001,"Prueba",10001));* por *listaarray.add(new Persona(10001,"Prueba",10001));* y *listalinked.add(5000,new Persona(10001,"Prueba",10001));* por *listalinked.add(new Persona(10001,"Prueba",10001));*

Obteniendo un resultado similar a este:

```
Tiempo invertido en insertar una persona en listaarray (en nanosegundos):
3800
Tiempo invertido en insertar una persona en listalinked (en nanosegundos):
1100
------------------------------------------------------------------------
BUILD SUCCESS
```

En donde vemos que los tiempos de inserción han disminuido en ambos casos tanto en la implementación con arrays como con listas enlazadas. En este último caso espectacularmente (en nuestro caso tan solo 1100 nanosegundos, aunque estos valores dependen del ordenador que estemos utilizando), lo que nos indica que LinkedList es muy eficiente en las inserciones en posición inicial y final de una lista.

CONCLUSIONES

Este ejemplo nos ha servido para sacar unas claras conclusiones ya no solo acerca del uso de ArrayList y LinkedList, sino sobre el uso de interfaces a modo general para saber qué implementación nos puede convenir utilizar en función del tipo de procesos que vayan a ser más habituales.

En cuanto a ArrayList y LinkedList vemos que las inserciones en posiciones iniciales o finales son por regla general más rápidas con LinkedList. Igualmente pasaría con las eliminaciones o borrados. Aunque observamos que las búsquedas o las inserciones en posiciones intermedias de una lista con gran número de elementos es más rápida en el caso de ArrayList. Esto es debido a que ésta accede a la posición de manera directa.

Para finalizar esta comparación vemos que el método add por defecto sin posición es más rápido que insertar en posiciones iniciales o intermedias en ambas implementaciones.

EJERCICIO

Crea una clase denominada Vehiculo con los atributos idVehiculo (int) y tipo (String), donde tipo podrá tomar los valores Coche, Camión, Furgoneta o Moto.

Crea una clase con el método main donde se introduzcan 5000 vehículos en una lista de tipo estático List. El atributo tipo debe establecerse para cada objeto de forma aleatoria. A continuación, el programa debe mostrar un resumen de cuántos vehículos hay de cada tipo. A continuación, debe recorrerse la lista y eliminarse todos los vehículos que no sean de tipo Coche. Por otro lado, deberán añadirse tantos vehículos de tipo Coche como se hayan eliminado, al final de la lista, de modo que los nuevos ids comenzarán a partir del último existente en la lista anterior. Además, deberá mostrarse de nuevo el resumen de cuántos vehículos hay de cada tipo y el tiempo empleado desde que comenzó la eliminación de elementos hasta que terminó la inserción de elementos. Responde a estas preguntas:

a) Implementa el programa usando ArrayList. ¿Cuál es el resultado que obtienes?

b) Implementa el programa usando LinkedList ¿Cuál es el resultado que obtienes?

c) Haz varias ejecuciones y compara los resultados. ¿Observas diferencias entre la ejecución con ArrayList y con LinkedList? Si observas diferencias, ¿cuáles son y a qué crees que se deben?

Ejemplo de ejecución:

Resumen lista inicial: hay 1189 Coches, 1357 Camiones, 1146 Furgonetas y 1308 Motos
Una vez realizada la eliminación e inserción:
Tiempo empleado en eliminación-inserción (en nanosegundos): 652854183
Resumen lista final: hay 5000 Coches, 0 Camiones, 0 Furgonetas y 0 Motos

Respuestas:

a) Implementando el programa usando ArrayList, se obtiene un resumen final de 2566 Coches y 0 vehículos de los otros tipos, con un tiempo empleado en eliminación-inserción de aproximadamente 4.8 milisegundos.

b) Implementando el programa usando LinkedList, se obtiene un resultado similar al de ArrayList, pero con posibles diferencias en los tiempos de ejecución debido a las diferencias en la estructura interna de datos.

c) Al hacer varias ejecuciones y comparar los resultados entre ArrayList y LinkedList, se pueden observar diferencias en los tiempos de ejecución, especialmente en la fase de eliminación e inserción de elementos, debido a las diferencias en la implementación subyacente de ambas estructuras de datos.