



INSTITUTO POLITECNICO NACIONAL



**UNIDAD INTERDISCIPLINARIA DE INGENIERIA
CAMPUS ZACATECAS**

DIFERENCIA ENTRE LIST, ARRAYLIST, LINKEFLIST

BRYAN ALEXIS GAYTAN MARTINEZ

2019670181

OSWALDO CRUZ LEIJA

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.

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.

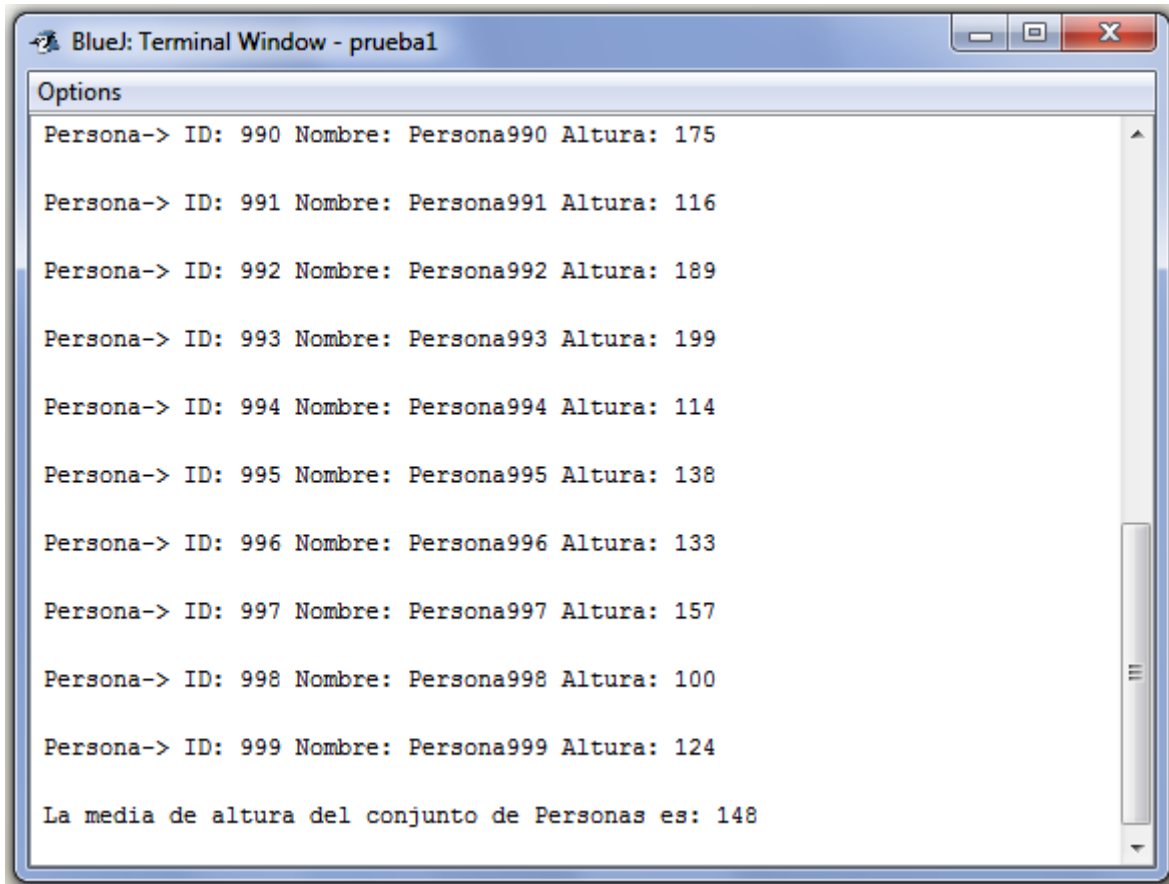
```
/* Ejemplo Interfaz List aprenderaprogramar.com */
public class Persona{
    private int idPersona;
    private String nombre;
    private int altura;

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

    public int getAltura() { return altura; } //Omitimos otros métodos get y set para
simplificar

    @Override
    public String toString() {
```

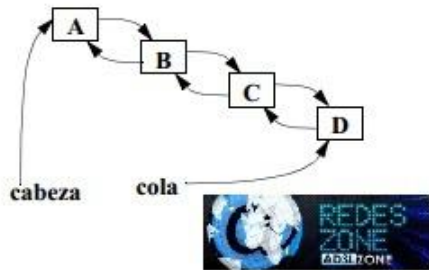
```
        return "Persona-> ID: "+idPersona+" Nombre: "+nombre+" Altura: "+altura+"\n";  
    }  
}
```



```
BlueJ: Terminal Window - prueba1  
Options  
Persona-> ID: 990 Nombre: Persona990 Altura: 175  
Persona-> ID: 991 Nombre: Persona991 Altura: 116  
Persona-> ID: 992 Nombre: Persona992 Altura: 189  
Persona-> ID: 993 Nombre: Persona993 Altura: 199  
Persona-> ID: 994 Nombre: Persona994 Altura: 114  
Persona-> ID: 995 Nombre: Persona995 Altura: 138  
Persona-> ID: 996 Nombre: Persona996 Altura: 133  
Persona-> ID: 997 Nombre: Persona997 Altura: 157  
Persona-> ID: 998 Nombre: Persona998 Altura: 100  
Persona-> ID: 999 Nombre: Persona999 Altura: 124  
La media de altura del conjunto de Personas es: 148
```

LinkedList: lista doblemente enlazada

- Acceso posicional costoso
- Inserción y extracción costosas
- Menos en la primera y última posición que es inmediato
- Tamaño ilimitado



Uso de ArrayList y LinkedList

Debe importarse el paquete java.util:

```
[java]import java.util.*;[/java]
```

Son clases genéricas

El constructor crea una lista vacía

```
[java]LinkedList<MiClase> lista=new LinkedList<MiClase> ();[/java]
```

Con las colecciones se puede usar el lazo for-each

```
[java]for(MiClase elem: lista) {
```

... utiliza elem ...

```
}[/java]
```

En las operaciones que se muestran en las transparencias siguientes la clase E es el parámetro genérico.

- Operaciones de modificación de la interfaz *List*

DIFERENCIAS ENTRE LIST Y ARRAYLIST:

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.

Mientras, un **ArrayList** es contenedor que contiene una implementación de la colección List.

Aquí puedes ver un ejemplo de las colecciones y sus relaciones en 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**.

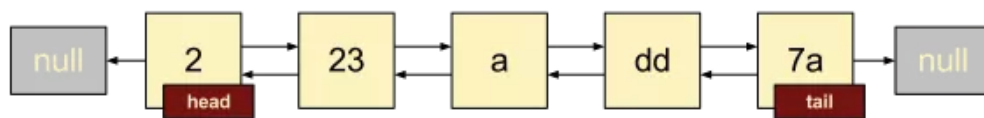
En tu duda, la **ArrayList** es un array que se maneja como una clase y no tiene tamaño fijo. Es eficiente cuando se realizan muchos acceso y en este caso los elementos se añaden al final. Permite que se puedan eliminar elementos intermedios, pero eso provocará un desplazamiento de índices.

En cambio, 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.

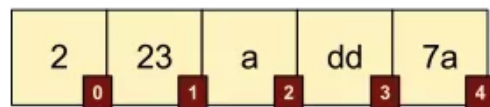
¿Qué sucede si utilizas **List** como lo haces en tu caso?, que podrías cambiar el tipo de **zzz** a **LinkedList** en su definición y todo sería compatible 100% y fácilmente. No obstante, en el primer caso de **xxx** tendrías que cambiar en todos los sitios el tipo debido a que estas implementando un **ArrayList** y podrías tener problemas si lo cambias a **LinkedList**. Por eso se recomienda el uso de **List**, como lo haces de **zzz**.

Array vs. Linked List

Linked List



Array



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)

Complejidad

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)

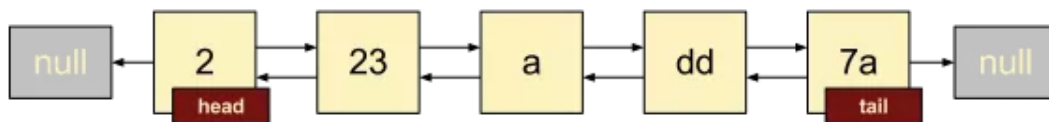
Gráfica comparativa

Bases para la comparación	Lista	Lista de arreglo
BASIC	La lista es una interfaz	ArrayList es una clase de colección estándar.
Sintaxis	Lista de interfaces	clase ArrayList
Extender / Implementar	La interfaz de lista amplía el Marco de Colección.	ArrayList extiende AbstractList e implementa List Interface.
Espacio de nombres	System.Collections.Generic.	Sistema.Colecciones.
Trabajo	Se utiliza para crear una lista de elementos (objetos) que están asociados con sus números de índice.	ArrayList se utiliza para crear una matriz dinámica que contiene objetos.

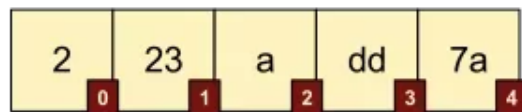
DIFERENCIAS ENTRE ARRAYLIST Y LINKEDLIST:

Array vs. Linked List

Linked List



Array



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.

Complejidad

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)	$O(n/2)$

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