**UPIIZ-IPN**

**“Unidad interdisciplinaria de Ingeniería**

**Campus Zacatecas”**

**Ing. Sistemas Computacionales.**

**Clase: Programación Orientada a Objetos**

**List, ArrayList, LinkedList.**

**Carlos Montes Leyva.  
Docente: Roberto Oswaldo Cruz Leija.**

**04/10/2019  
Zacatecas, Zacatecas, México**

# List:

[1]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.

# Array list:

[5]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:

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";

    }

}

import java.util.List;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.Random;

public class **Programa** {

    public static void **main**(String arg[]) {

            List<Persona> lp = new ArrayList<Persona>(); // El tipo es List y lo implementamos con ArrayList

            Random r = new Random();

            Persona temp = null;

            int sumaaltura = 0;

            for(int i=0;i<1000;i++)

            {lp.add(new Persona(i,"Persona"+i,r.nextInt(100)+100));}

            Iterator<Persona> it = lp.iterator();

            while(it.hasNext())

            {

                temp = it.next();

                System.out.println(temp);

                sumaaltura += temp.getAltura();

            }

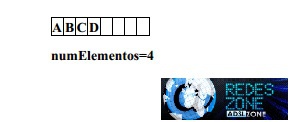
            System.out.println("La media de altura del conjunto de Personas es: "+sumaaltura/lp.size());

      }

}

**[2]ArrayList: lista implementada con un array**

* Acceso posicional eficiente
* Inserción y extracción costosas menos en la última posición que es instantánea
* Cuando se supera el tamaño del array, se crea uno nuevo más grande y se copian en él los elementos del antiguo

**[](https://www.redeszone.net/content/uploads/cursoJava_ArrayList_1.jpg)**

# LinkedList:

[3]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.

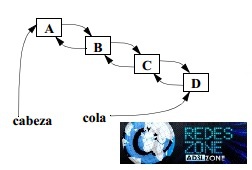
Ahora centrándonos en la clase LinkedList, ésta se basa en la implementación de listas doblemente enlazadas. Esto quiere decir que la estructura es un poco más compleja que la implementación con ArrayList, pero… ¿Qué beneficios nos aporta si la estructura es más compleja?.

¿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).

**[5]LinkedList: lista doblemente enlazada**

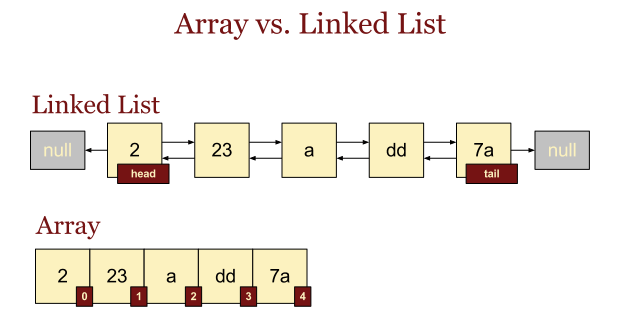
* Acceso posicional costoso
* Inserción y extracción costosas
* Menos en la primera y última posición que es inmediato
* Tamaño ilimitado



# LinkedList y ArrayList:

[5]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.

# Referencias:

[1]<https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=603:interface-list-del-api-java-clases-arraylist-linkedlist-stack-vector-ejemplo-con-arraylist-cu00920c&catid=58&Itemid=180>

[2] <https://www.redeszone.net/2012/03/05/curso-de-java-estructuras-de-datos-arraylist-y-linkedlist/>

[3] <https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=605:interface-list-clase-linkedlist-api-java-ejercicio-diferencias-entre-arraylist-y-linkedlist-codigo-cu00921c&catid=58&Itemid=180>

[4] <http://www.enrique7mc.com/2016/07/diferencia-entre-arraylist-y-linkedlist/>