C:\Users\belen.moreno\Documents\NetBeansProjects\AplicaciónListaArray
\src\aplicacionlistaarray\ListaArray.java

```
package aplicacionlistaarray;
import java.util.*; // importamos esta librería porque en ella se encuentra
                    // definida la excepción NoSuchElementException.

/**-------------------------------------------------------------------------
 * CLASE ListaArray QUE IMPLEMENTA A LA INTERFAZ Lista.java MEDIANTE UN ARRAY
 * UTILIZANDO PARA LOS DATOS UN ARRAY UNIDIMENSIONAL Y UNA VARIABLE LONGITUD.
 *
 * Esta implementación es muy eficiente cuando las inserciones y supresiones se
 * realizan a través de un índice (lista indexada).
 * @author bmoreno
 * -------------------------------------------------------------------------*/
public class ListaArray implements Lista {

    //datos:
    private int numElementos = 5;
    int lista[] = new int[numElementos];
    int longitud; //contiene el número de elementos actual de la lista.

    /**-----------------------------------------------------------------
     * Constructor que inicializa los datos del array y la longitud a cero
     * Complejidad O(N)
     */
        public ListaArray(){
        longitud = 0;
        for (int i = 0; i < numElementos; i++)
            lista[i] = 0;
        }

    /**-------------------------------------------------------------------------
     * @return. Devuelve el número de elementos actual de la lista (longitud).
     * Complejidad O(1)
     */
        @Override
    public int size(){
        return longitud;
    }

    /**-----------------------------------------------------------------------
     * Devuelve true si la lista está vacía, false en c.c.
     * Complejidad O(1)
     * @return
     */
        @Override
    public boolean isEmpty(){
    boolean vacia;

        if (longitud == 0)
            vacia = true;
        else
            vacia = false;

        return vacia;
    }

    /**-------------------------------------------------------------
     * Returns the element at the specified position in this list.
     * @param index, Requisito: index debe cumplir 0 <= index < longitud. Esto
     * debe comprobarse antes de la llamada. Si no lo cumple lanza una excepción.
     * @return. Devuelve el elemento que hay en la lista en la posición index,
     * donde index es un índice que vale 0 para el primer elemento, 1 para el
     * segundo...
     * Complejidad: O(1)
     */
        @Override
    public int get(int index) throws IndexOutOfBoundsException {

        if ((!isEmpty()) && (index >= 0 && index < longitud))
                return lista[index];
        else
            throw new IndexOutOfBoundsException("Error en get(): argumento inválido");
```

```
}

/**-------------------------------------------------------------------
 * Replaces the element at the specified position in this list with the
 * specified element elem. Returns the original element.
 * @param index. Requisito: debe cumplir 0<= index < longitud. Si esto no se
 * cumple no puede hacerse la llamada a este método, pues daría un error.
 * @param elem
 * @return Asigna elem a la posición index y devuelve el elemento que había
 * en esa posición anteriormente.
 * Complejidad: O(1)
 */
    @Override
public int set(int index, int elem) throws IndexOutOfBoundsException {

    int antiguo;

    if ((!isEmpty()) && (index >= 0 && index < longitud)){
          antiguo = lista[index];
          lista[index] = elem;
          return antiguo;
    }
    else
        throw new IndexOutOfBoundsException("Error en set(): argumento inválido");
}

 /**-------------------------------------------------------------------
 * Inserts the element elem at the specified position in this list.
 * Inserta un nuevo elemento elem en la posición index si
// Complejidad O(N)
 * @param index. Requisito: debe cumplir 0 <= index < longitud. Si no lo
 * cumple lanzará una excepción.
 * @param elem
 */
    @Override
public void add(int index, int elem) throws IndexOutOfBoundsException {

if ((index < 0) || (index >= longitud))
    throw new  IndexOutOfBoundsException("Error en add(): argumento inválido");
    else {
    if (longitud == numElementos){ //Array lleno.Se duplica su longitud.
        numElementos = numElementos*2;

        int[] A = new int[numElementos];
        for (int j=0; j < longitud; j++)
            A[j] = lista[j];
        lista = A;
    } // end if

    for (int j = longitud - 1; j >= index ; j--)
        lista[j+1] = lista[j];
    lista[index] = elem;
    longitud++;
    } // end else
} // end add


/**-------------------------------------------------------------------------
 * Borra y devuelve el elemento elem de índice i  y desplaza los
 * demás elementos una posicion hacia el principio.
 * Complejidad O(N)
 * @param index. Requisito: debe cumplir: 0 <= index < longitud. Si no lo
 * cumple lanza una excepción.
 * @return
 * O(N)
 */
    @Override

public int remove(int index) throws IndexOutOfBoundsException{

    if (isEmpty() || (index < 0) || (index >= longitud))
        throw new IndexOutOfBoundsException("Error en remove(): arg. inválido");
    else{
        int temp = lista[index];
```

```
            for (int j = index; j < longitud-1; j++)
                lista[j] = lista[j+1];
            longitud--;

            return temp;
        }
    }

    /**-------------------------------------------------------------------------
     * Inserta elem al final de la lista
     * Complejidad O(N). Sería O(1) si el numElementos fuese cte y no se duplicase.
     * @param elem
     * @return true si hay inserción, false en caso contrario ( si no se
     * permitiesen elementos repetidos.)
     */
        @Override
    public boolean add(int elem){

        boolean resultado = false;

        if (longitud == numElementos){
            //duplicamos el tamaño del array
            numElementos = numElementos*2;

            int[] A = new int[numElementos];

            for (int i=0; i < longitud; i++)
                A[i] = lista[i];
            lista = A;
        }
        lista[longitud] = elem;
        longitud++;
        resultado = true;

    return resultado;
    }


    /**-------------------------------------------------------------
     * Returns true if this list contains the specified element elem.
     * @param elem
     * @return
     * O(N)
     */
    @Override
    public boolean contains(int elem){

        boolean encontrado = false;

        for (int i=0; i < longitud; i++){
            if (elem == lista[i])
                encontrado = true;
        }
        return encontrado;
    }

    /**-------------------------------------------------------------
     * Devuelve un String con los valores de la lista y su longitud.
     * Complejidad O(N).
     * @return
     */
        @Override   //sobreescribe al metodo de java.lang.Object
    public String toString(){
        String cadena = "[";
        for (int i = 0; i < longitud; i++){
            cadena = cadena + lista[i] + " ";
        }
        cadena = cadena + "]" + "longitud = " + longitud;
        return cadena;
    }


     /**-------------------------------------------------------------------------
     * Recibe un elem. y elimina su 1ª aparición en la lista. Removes the first
     * occurrence of the specified element elem from this list, if it is present.
     * Returns true if elem has been removed, false otherwise.
```

```
     * @param elem
     * @return  Devuelve true si hay eliminación, false si no se encuentra elem
     * en la lista.
     * Complejidad O(N).
     */
        @Override
    public boolean removeElem(int elem) {

    boolean encontrado = false;

    int i = 0;
    while ((i < longitud) && (!encontrado)){
        if (lista[i] == elem){
            encontrado = true;
            for (int j = i+1; j < longitud; j++)
                lista[j-1] = lista[j];
            longitud--;
        }
        i++;
    }
    return encontrado;
    }

    /**-------------------------------------------------------
     * Borra todos los elementos de la lista dejándola vacía
     * O(1)
     */
    @Override
    public void clear() {
        longitud = 0;
    }

    /**-----------------------------------------------------------------
     * Calcula el índice del elemento elem en la lista.
     * @param elem
     * @return the index of the first occurrence of the specified element
     * in this list, or -1 if this list does not contain the element.
     * O(1)
     */
    @Override
    public int indexOf(int elem) {
        int pos = -1;

        for (int i = 0; i < longitud; i ++)
            if (lista[i] == elem)
                pos = i;
        return pos;
    }

    /**--------------------------------------------------------------------
     * Compara la lista invocante con la lista l y devuelve true si son iguales.
     * @param l
     * @return
     * O(N)
     */
    @Override
    public boolean equals(Lista l) {
        boolean iguales = true;

        if (l.size() != size())
            iguales = false;
        else
            for (int i = 0; i < longitud; i++)
                if (lista[i] != l.get(i))
                    iguales = false;

        return iguales;
    }

    // NOTA: los siguientes métodos no están en el interface Lista.
    // Son adecuados en implementaciones con listas enlazadas donde las
    // todas estas operaciones son O(1)

    /**-------------------------------------------------------------------
     * Devuelve el primer elemento de la lista
     * Requisito: Debe cumplirse que la lista no esté vacía (esto se comprueba
```

```
     * antes de la llamada)
     * O(1)
     * @return
     * @throws NoSuchElementException
     */
    public int getFirst() throws NoSuchElementException {
       // La excepción NoSuchElementException se encuentra definida en la
       // librería java.util, por tanto, para que no de error es preciso
       // importar dicha librería.

       if (isEmpty())
            throw new NoSuchElementException("Error en getFirst: la lista está vacía");
       else
            return lista[0];
    }

    /**-----------------------------------------------------------------
     * Devuelve el último elemento de la lista
     * Requisito: la lista no esté vacía (esto se comprueba antes de la
     * llamada)
     * O(1)
     * @return
     * @throws NoSuchElementException
     */
    public int getLast() throws NoSuchElementException{
       if (isEmpty())
               throw new NoSuchElementException("Error en getLast: la lista está vacía");
       else
            return lista[longitud-1];
    }

      /**-----------------------------------------------------------------------
     * Requisito: la lista no esté vacía.
     * Elimina y devuelve el 1º elemento de la lista si lo hay. Produce una
     * excepción si no tiene elementos y por tanto no hay eliminacion.
     * Removes and returns the first element from this list Complejidad O(N).
     * @return
     * @throws NoSuchElementException
     * O(N)
     */
    public int removeFirst () throws NoSuchElementException {

    int resultado = lista[0];

    if (isEmpty())
               throw new NoSuchElementException("Error en removeFirst: la "
                       + "lista está vacía");
       else {
        for (int i = 1; i < longitud-1; i++)
            lista[i] = lista[i+1];
        longitud--;
        return resultado;
        }
    }


     /**-----------------------------------------------------------------------
     * Requisito: la lista no está vacía.
     * Elimina y devuelve el último elemento de la lista si lo hay. Produce una
     * excepción si no tiene elementos y por tanto no hay eliminacion.
     * Removes and returns the último element from this list
     * Complejidad O(N).
     * @return
     * @throws NoSuchElementException
     * O(N)
     */
    public int removeLast () throws NoSuchElementException {

    if (isEmpty())
               throw (new NoSuchElementException("Error en removeLast: "
                       + "la lista está vacía"));
       else {
                int resultado = lista[longitud-1];

                longitud--;
```

```
                return resultado;
            }
        }

         /**-------------------------------------------------------------------
         * Inserta elem al principio de lista. Si lista esta llena, dobla su tamaño.
         * Complejidad O(N)
         * @param elem
         */
            public void addFirst(int elem) {

            if (longitud == numElementos){ //Array lleno.Se duplica su longitud.
                numElementos = numElementos*2;

                int[] A = new int[numElementos];
                for (int i=0; i < longitud; i++)
                    A[i] = lista[i];
                lista = A;
                }
            //desplazamos los elem. una pos hacia el final
                for (int i = longitud -1; i >= 0; i-- )
                    lista[i+1] = lista[i];

            lista[0] = elem;
            longitud++;
        }

         /**-------------------------------------------------------------------
         * Inserta elem al final de lista. Si lista esta llena, dobla su tamaño.
         * Complejidad O(N) si se redimensiona el array, O(1) si no.
         * @param elem
         */
            public void addLast(int elem) {

            if (longitud == numElementos){ //Array lleno.Se duplica su longitud.
                numElementos = numElementos*2;

                int[] A = new int[numElementos];
                for (int i=0; i < longitud; i++)
                    A[i] = lista[i];
                lista = A;
                }

                //desplazamos los elem. una pos hacia el final

            lista[longitud] = elem;
            longitud++;
        }

        /** -------------------------------------------------------------------
          * Devuelve un objeto iterador (de la clase Iterator) sobre los elementos de
          * esta lista. Un objeto iterador permite al usuario de la lista hacer un
          * recorrido por sus elementos desde el primer al último, pasando por cada
          * uno una sóla vez, con objeto de usar cada elemento en alguna expresión
          * o escribirlo en la pantalla, etc. pero sin modificarlo ni borrarlo,
          * es decir, el iterador da los valores de los elementos de la lista
          * pero sin modificar la lista.
          * @return. iterador sobre los elementos de esta lista.
          * O(1)
          */
            @Override
         public Iterator iterator(){

         Iterator it = new Iterator(this);
         return it;
    }

    }
```