Tema 5. Estucturas de datos - ArrayList

1. INTRODUCCIÓN

Un ArrayList es una estructura de datos dinámica que sirve para almacenar objetos de cualquier tipo.

Al ser una estructura de datos dinámica, cuando se crea no es necesario darle un tamaño inicial. El tamaño de un ArrayList aumenta según nuestras necesidades y puede modificarse (aumentar o disminuir) durante la ejecución del programa.

Para utilizar la clase ArrayList es necesario importarla: `import java.util.ArrayList;`

2. DECLARACIÓN Y CREACIÓN DE ARRAYLIST

De forma general un Arraylist se crea de esta forma: `ArrayList A = new ArrayList();`

Esta instrucción crea un ArrayList llamado A.

Para añadir objetos a un ArrayList se utiliza el método **add().** Por ejemplo:

```
A.add("Lenguaje");
A.add(3);
A.add('a');
A.add(23.5);
```

Después de estas instrucciones el contenido del ArrayList es el siguiente:

"Lenguaje"	3	'a'	23.5
0	1	2	3

En un ArrayList declarado así podemos añadir objetos de cualquier tipo.

A diferencia de los arrays que hemos visto hasta ahora, **un ArrayList solo puede contener objetos**.

En este ejemplo, el primer objeto que se añade es el String “Lenguaje”. Un String es un objeto.

Pero el resto de elementos que se han añadido no son objetos, son datos de tipo básico:

3 de tipo int, ‘a’ de tipo char y 23.5 de tipo double.

Como ArrayList solo puede almacenar objetos, para solucionar este problema y permitir que se puedan guardar datos de tipo básico, **Java convierte los tipos básicos en objetos** antes de añadirlos al ArrayList. Este proceso de conversión es transparente al programador y no tenemos que indicar nada para que se realice. Java realiza la conversión automáticamente.

Hay estructuras de datos como ArrayList que solamente manipulan objetos. Para poder utilizar tipos primitivos con estas estructuras de datos, Java provee las llamadas clases envolventes también llamadas **Clases Contenedoras o Wrappers**. Cada tipo primitivo tiene su correspondiente clase envolvente:

Tipo primitivo	Clase envolvente
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Para añadir un dato de tipo primitivo al ArrayList, Java realiza antes la conversión del tipo primitivo a su clase envolvente. Por ejemplo, cuando se añade un int al ArrayList se convierte antes a Integer.

ArrayList con elementos del mismo tipo

Aunque podamos almacenar en un mismo ArrayList objetos que no son del mismo tipo, trabajar con este tipo de estructuras puede tener alguna complicación.

Por ejemplo, si queremos realizar operaciones solo con los datos numéricos que contiene, cuando se recorre el ArrayList tendremos que estar siempre comprobando si el elemento es un valor numérico o no antes de realizar la operación.

La forma más habitual de trabajar es utilizando ArrayList cuyos objetos son todos del mismo tipo.

Cuando se crea el ArrayList se indica el tipo de objetos que contendrá.

La forma general de crear un ArrayList indicando su tipo es esta:

```
ArrayList<Tipo> nombreArray = new ArrayList<>();
```

En **<Tipo>** se indica el tipo de objetos que contiene el ArrayList.

Como un ArrayList solo puede contener objetos, no podemos indicar en <Tipo> tipos de datos primitivos (int, double, char, etc). Tendremos que usar su clase envolvente (Integer, Double, Character, etc).

Por ejemplo, para crear un ArrayList llamado *enteros* que va a almacenar números de tipo entero escribe la instrucción:

```
ArrayList<Integer> enteros = new ArrayList<>();
```

Para crear un ArrayList llamado *precios* para almacenar el precio de venta de un artículo se escribe la instrucción:

```
ArrayList<Double> precios = new ArrayList<>();
```

Para crear un ArrayList llamado *nombres* que va a almacenar nombres de personas (objetos de String) se escribe la instrucción:

```
ArrayList<String> nombres = new ArrayList<>();
```

En las primeras versiones de Java era necesario repetir de nuevo el tipo de la colección al crearla.

Por ejemplo, para crear un ArrayList de String había que escribir:

```
ArrayList<String> nombres = new ArrayList<String>();
```

A partir de Java 7 dejó de ser necesario indicar el tipo de los elementos al crear la colección.

```
ArrayList<String> nombres = new ArrayList<>();
```

El tipo de la colección se indica en la parte izquierda de la instrucción y Java es capaz de inferir el tipo de los elementos de la colección que está creando a partir de ese tipo.

Los símbolos <> en colecciones se llaman **operador diamante**.

El operador diamante permite inferir el tipo de la colección que se crea sin necesidad de indicarlo expresamente.

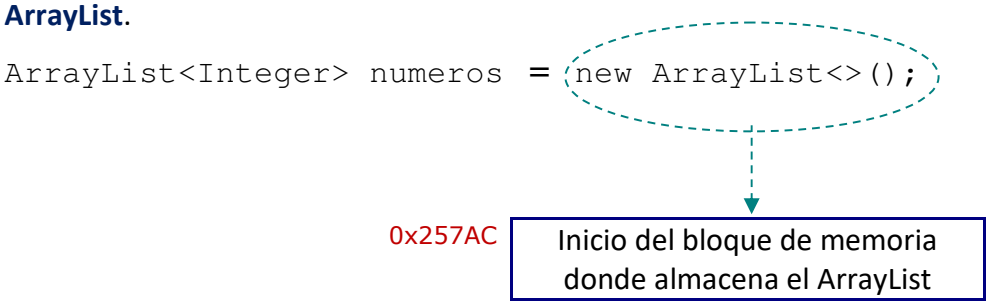
Almacenamiento en memoria del ArrayList

Cuando se crea un ArrayList, por ejemplo:

```
ArrayList<Integer> numeros = new ArrayList<>();
```

Con esa instrucción se crea un ArrayList para guardar enteros que inicialmente está vacío.

Igual que ocurre con los arrays que hemos visto hasta ahora, cuando se crea un ArrayList, **el nombre del ArrayList contiene la referencia o dirección de inicio del bloque de memoria donde se almacena el ArrayList**.



A la variable `numeros` se le asigna la dirección de memoria dond e se encuentra el ArrayList.

`numeros = 0x257AC`

3. ALGUNOS MÉTODOS DE ARRAYLIST

MÉTODO	DESCRIPCIÓN
size()	Devuelve el número de elementos (int)
add(X)	Añade el objeto X al final.
add(posición, X)	Inserta el objeto X en la posición indicada.
get(posicion)	Devuelve el elemento que está en la posición indicada.
remove(posicion)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
remove(X)	Elimina la primera ocurrencia del objeto X. Devuelve true si el elemento está en la lista.
clear()	Elimina todos los elementos.
set(posición, X)	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
contains(X)	Comprueba si la colección contiene al objeto X. Devuelve true o false.
indexOf(X)	Devuelve la primera posición donde se encuentre el objeto X. Si no se encuentra devuelve -1
lastIndexOf(X)	Devuelve la última posición donde se encuentre el objeto X. Si no se encuentra devuelve -1

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ArrayList.html

4. RECORRIDO DE UN ARRAYLIST

Con un **bucle for**

```
for(int i = 0; i < array.size(); i++){
    System.out.println(array.get(i));
}
```

Con un **bucle for-each** o bucle for mejorado

Si suponemos un ArrayList de números enteros*:*

```
for(Integer i: numeros){
    System.out.println(i);
}
```

Si el ArrayList contiene objetos de tipos distintos o desconocemos el tipo:

```
for(Object o: nombreArray){
    System.out.println(o);
}
```

Utilizando un **objeto Iterator**.

Para utilizar un iterador hay que importarlo: `import java.util.Iterator;`

Para recorrer un Arraylist con un iterador podemos utilizar los métodos:

hasNext(): devuelve true si hay más elementos en el array.

next(): devuelve el siguiente objeto contenido en el array.

Ejemplo:

```
ArrayList<Integer> numeros = new ArrayList<>();
numeros.add(1);
numeros.add(10);
numeros.add(-1);
numeros.add(31);
//se crea el iterador it para el ArrayList numeros
Iterator it = numeros.iterator();
while(it.hasNext()){                    //mientras queden elementos
        System.out.println(it.next());  //se obtienen y se muestran
}
```

Una de las ventajas de utilizar un Iterator es que no necesitamos indicar el tipo de objetos que contiene el ArrayList si solamente los vamos a usar para poder recorrerlo y mostrar el contenido.

Sin embargo, hay ocasiones en las que si es necesario indicar el tipo al crear el Iterador.

Por ejemplo, si queremos sumar los elementos del ArrayList *números* del ejemplo anterior:

```
int suma = 0;
//se crea el iterador it para el ArrayList numeros
Iterator it = numeros.iterator();
while(it.hasNext()){                    //mientras queden elementos
      suma = suma + it.next();          //se obtiene un elemento y se suma
}
System.out.println(suma);
```

En este ejemplo el compilador nos indicará un error en la instrucción:

```
suma = suma + it.next();
```

Como el iterador se ha creado sin tipo, **el método it.next() devuelve un dato de tipo Object**, que no se puede sumar a un tipo int.

En estos casos la mejor solución es indicar el tipo de dato que devuelve el operador cuando se crea:

```
//se crea el iterador it de tipo Integer para el ArrayList numeros
Iterator<Integer> it = numeros.iterator();
while(it.hasNext()){                    //mientras queden elementos
      suma = suma + it.next();          //se obtiene un elemento y se suma
}
System.out.println(suma);
```

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Iterator.html

Utilizando un **ListIterator.**

Para utilizar un ListIterator hay que importarlo: `import java.util.ListIterator;`

Para recorrer y acceder a los elementos de la colección se utilizan los métodos:

hasNext(): devuelve true si hay más objetos en la colección.

hasPrevious(): devuelve true si hay más objetos en la colección. Se utiliza si estamos recorriendo la colección en orden inverso.

next(): devuelve el siguiente objeto contenido en la colección.

previous(): devuelve el siguiente objeto contenido en la colección. Se utiliza si estamos recorriendo la colección en orden inverso.

Mediante un ListIterator podemos recorrer solo una parte de la colección indicando la posición de inicio. Para recorrerlo completamente en orden inverso indicaremos como posición inicial la última del ArrayList.

Ejemplo:

```
ArrayList<String> nombres = new ArrayList<>();
nombres.add("Andrea");
nombres.add("Lucas");
nombres.add("Juan");
nombres.add("Francisco");
System.out.println("Contenido del ArrayList:");
//recorrido de principio a fin utilizando un ListIterator
ListIterator<String> it = nombres.listIterator();
while (it.hasNext()) {
	System.out.println(it.next());
}

//recorrido desde la posición 2 hasta el final
System.out.println("Contenido a partir del segundo elemento:");
ListIterator<String> it1 = nombres.listIterator(2); // indicamos la posición de inicio
while (it1.hasNext()) {
	System.out.println(it1.next());
}

//recorrido del ArrayList en orden inverso utilizando ListIterator
//indicamos que la posición inicial es la última y recorremos con los
//métodos hasPrevious() y previous()
System.out.println("Recorrido en orden inverso");
ListIterator<String> it2 = nombres.listIterator(nombres.size());
while (it2.hasPrevious()) {
	System.out.println(it2.previous());
}
```

Salida por pantalla:

```
Contenido del ArrayList:
Andrea
Lucas
Juan
Francisco
Contenido a partir del segundo elemento:
Juan
Francisco
Recorrido en orden inverso
Francisco
Juan
Lucas
Andrea
```

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ListIterator.html

Con el **método forEach**.

Java 8 incorporó un nuevo método para recorrer colecciones: el método forEach. Este método utiliza expresiones lambda y referencias a métodos, también incorporados al lenguaje en la versión Java 8, que simplifican las operaciones con las colecciones.

Ejemplo:

```
ArrayList<String> nombres = new ArrayList<>();
nombres.add("Andrea");
nombres.add("Lucas");
nombres.add("Juan");
nombres.add("Francisco");
nombres.forEach(x -> System.out.println(x));
```

Salida por pantalla:

```
Andrea
Lucas
Juan
Francisco
```

En la instrucción: `nombres.forEach(s -> System.out.println(s));`

El método forEach recorre el ArrayList nombres y para cada elemento lo muestra por pantalla.

`s -> System.out.println(s)` es una expresión lambda.

Las expresiones lambda se verán al final de este curso pero son muy utilizadas en las colecciones porque facilitan bastante el trabajo y por eso se muestran aquí.

Aunque no se han explicado, de forma intuitiva se puede entender lo que significa. De forma muy resumida, una expresión lambda indica una acción a realizar. En este ejemplo la acción a realizar es mostrar por pantalla cada objeto del ArrayList. La s hace referencia a cada objeto de la colección y a continuación de la flecha `->` estamos indicando lo que hay que hacer con ese objeto s, en este caso mostrarlo por pantalla. Lo hemos llamado s pero podemos darle cualquier otro nombre.

El método forEach recorre el ArrayList y para cada objeto realiza la acción indicada. En este caso para cada String lo muestra por pantalla. Esa variable s es la equivalente a la que aparece en el bucle for-each que ya conocemos:

```
for(String s: nombres){
    System.out.println(s);
}
```

En el ejemplo se ha utilizado en el método forEach una expresión lambda pero también se puede utilizar un método referenciado.

Ejemplo:

```
ArrayList<String> nombres = new ArrayList<>();
nombres.add("Andrea");
nombres.add("Lucas");
nombres.add("Juan");
nombres.add("Francisco");
nombres.forEach(System.out::println);
```

Salida por pantalla:

```
Andrea
Lucas
Juan
Francisco
```

En la instrucción `System.out::println` se está referenciando al método println. Podemos usar referencias a métodos en lugar de expresiones lambda y así hacer el código más simple.

En el ejemplo la referencia al método println le indica al método forEach lo que debe hacer con cada elemento de la colección, en este caso mostrarlo por pantalla.

El operador `::` se usa para indicar referencia a método.

5. EJEMPLOS DE USO DE ARRAYLIST

Ejemplo 1:

Uso básico de un ArrayList

Crear un ArrayList, añadir elementos, insertar elementos en una posición determinada, eliminar elementos, modificar un elemento, obtener el elemento que se uncuentra en una posición determinada, mostrar todo el contenido del ArrayList.

```
//se crea un ArrayList de tipo String
ArrayList<String> nombres = new ArrayList<>();

//se añaden tres elementos al ArrayList mediante el método add().
//Los elementos se añaden al final del ArrayList.
nombres.add("Ana");
nombres.add("Luisa");
nombres.add("Felipe");

//se muestra el contenido actual de ArrayList
System.out.println(nombres); //Muestra -> [Ana, Luisa, Felipe]

//se inserta el String "Pablo" en la posición 1 del ArrayList
nombres.add(1, "Pablo");

//se muestra el contenido actual de ArrayList
System.out.println(nombres); //Muestra -> [Ana, Pablo, Luisa, Felipe]

//se elimina el elemento que se encuentra en la posición 0
nombres.remove(0);

//se muestra el contenido actual de ArrayList
System.out.println(nombres); //Muestra -> [Pablo, Luisa, Felipe]

//se modifica el contenido del elemento 0 del ArrayList
nombres.set(0,"Alfonso");

//se muestra el contenido actual de ArrayList
System.out.println(nombres); //Muestra -> [Alfonso, Luisa, Felipe]

//se asigna al String s1 el valor del elemento que se encuentra en la posición 1
String s1 = nombres.get(1);

//se asigna al String s2 el valor del elemento que se encuentra en la última
//posición del ArrayList
String s2 = nombres.get(nombres.size() - 1);

//se muestra el contenido de las variables s1 y s2
System.out.println(s1 + " " + s2); //Muestra -> Luisa Felipe
```

A diferencia de lo que ocurre con los arrays, podemos mostrar el contenido de un ArrayList utilizando print o println.

```
System.out.println(nombres);
```

Muestra todo el contenido del ArrayList con los elementos separados por comas. Además los elementos se muestran entre corchetes [ ] .

Ejemplo 2:

Programa que lee números enteros hasta que se lee un 0 y los guarda en un ArrayList.

A continuación se mostrarán todos los números introducidos excepto el 0.

Los números se mostrarán uno por línea.

Se mostrará también la suma y la media de los números introducidos y cuántos números hay superiores a la media. La media se mostrará con dos decimales.

Se utilizará un iterador para recorrer el ArrayList y mostrar su contenido, un bucle for para sumar los elementos y calcular la media y un bucle for-each para calcular cuántos números son superiores media.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;
public class ArrayList2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<Integer> numeros = new ArrayList<>();
        int n, contador = 0;
        double suma = 0, media;
        do {
            System.out.println("Introduce números enteros. 0 para acabar: ");
            System.out.print("Número: ");
            n = sc.nextInt();
            if (n != 0) {
                numeros.add(n);
            }
        } while (n != 0);

        System.out.println("Valores introducidos:");

        //Mostrar el contenido del ArrayList
        //Se utiliza un iterador para recorrerlo tal y como pide el enunciado
        Iterator it = numeros.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }

        //recorrido del ArrayList para calcular la media
        //se utiliza un bucle for
        for (int i = 0; i < numeros.size(); i++) {
            suma += numeros.get(i);
        }
        media = suma / numeros.size();

        //recorrido para calcular cuantos elementos son superiores a la media
        //se utiliza un bucle for-each
        for (Integer i : numeros) {
            if (i > media) {
                contador++;
            }
        }

        System.out.println("Suma: " + suma);
        System.out.printf("Media: %.2f%n", media);
        System.out.println("Hay " + contador + " números mayores que la media");
    }
}
```

6. COPIAR UN ARRAYLIST

Disponemos de un ArrayList de enteros llamado ventas:

Si queremos hacer una copia de este ArrayList, en principio podemos pensar que la siguiente instrucción hace la copia:

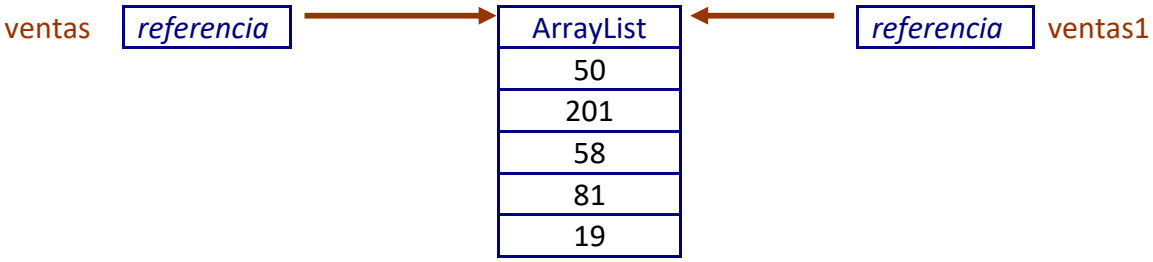
```
ArrayList<Integer> ventas1 = ventas;
```

pero **esta instrucción no crea una copia del ArrayList** *ventas* sino que está asignando a la variable *ventas1* el contenido de la variable *ventas*.

Como la variable *ventas* contiene la dirección de memoria del ArrayList, lo que estamos asignando a *ventas1* es la dirección de memoria del ArrayList.

Lo que se ha creado es un *alias* del ArrayList.

Ahora tenemos dos formas de acceder al ArrayList: mediante la variable ventas y mediante la variable ventas1.



Para hacer una copia de un Arraylist podemos crear un ArrayList vacío y mediante un bucle ir añadiendo los elementos del ArrayList original en el nuevo o se puede hacer de esta forma:

```
ArrayList<Integer> ventas1 = new ArrayList<>(ventas);
```

Esta instrucción crea un nuevo ArrayList llamado *ventas1* y copia el contenido del ArrayList *ventas*. Ahora tendremos dos ArrayList distintos con el mismo contenido.

7. ARRAYLIST Y MÉTODOS

Un ArrayList puede ser pasado como parámetro a un método.

Además un método puede devolver un ArrayList mediante la sentencia return.

Ejemplo: Programa que pide por teclado números enteros y los guarda en un ArrayList. Se eliminan del ArrayList los valores menores o iguales a cero y se muestra el ArrayList modificado.

Se utilizarán los siguientes métodos:

- Método para leer por teclado los valores y añadirlos al ArrayList.
- Método para mostrar por pantalla un ArrayList de enteros.
- Método para eliminar los valores menores o iguales a cero de un ArrayList de enteros.

```
import java.util.ArrayList;
import java.util.Scanner;
public class EliminarValores {
    public static void main(String[] args) {
        ArrayList<Integer> positivos = new ArrayList<>();
        leerArrayList(positivos);
        mostrarArrayList(positivos);  //se muestra el ArrayList original
        modificarArrayList(positivos);
        mostrarArrayList(positivos);  //se muestra el ArrayList modificado
    }
    //método para añadir valores al ArrayList. Recibe como parámetro el ArrayList
    //de enteros donde vamos a guardar los números introducidos por teclado
    public static void leerArrayList(ArrayList<Integer> A) {
        Scanner sc = new Scanner(System.in);
        int N, numero;
        do {
            System.out.print("Cuantos números vas a introducir? ");
            N = sc.nextInt();
        } while (N <= 0);
        for (int i = 1; i <= N; i++) {
            System.out.print("Numero " + i + ": ");
            numero = sc.nextInt();
            A.add(numero);
        }
    }
    //método para mostrar un ArrayList de enteros por pantalla
    public static void mostrarArrayList(ArrayList<Integer> a) {
        System.out.println(a);
    }
    //método que recibe un ArrayList de enteros y elimina todos los elementos <= 0
    public static void modificarArrayList(ArrayList<Integer> a) {
        for(int i = 0; i<a.size(); i++){
            if(a.get(i)<=0){
                a.remove(i);
                i--;
            }
        }
    }
}
```

Cuando se pasa un ArrayList a un método como parámetro estamos pasando al método la dirección de memoria donde se encuentra el ArrayList por lo tanto cualquier modificación que se realice en el ArrayList dentro del método la estamos haciendo sobre el ArrayList original. Por eso no es necesario devolver mediante return el ArrayList desde el método.

Ejemplo: Programa que lee nombres por teclado y los guarda en un ArrayList. La lectura de nombres finaliza cuando se responda "NO" a la pregunta *Continuar leyendo? (SI/NO).* A partir de este ArrayList el programa creará otro con el contenido invertido. Se mostrará por pantalla el ArrayList original y nuevo ArrayList con el contenido invertido.

Se utilizarán los siguientes métodos:

- Método para leer por teclado los valores y añadirlos al ArrayList.
- Método para mostrar los elementos de un ArrayList de String, uno por línea.
- Método para invertir y devolver el contenido de un ArrayList de String.

```
import java.util.ArrayList;
import java.util.Scanner;

public class ArrayList4 {

    public static void main(String[] args) {
        ArrayList<String> nombres = new ArrayList<>();
        leer(nombres);
        mostrar(nombres);
        ArrayList<String> invertido = invertir(nombres);
        mostrar(invertido);
    }

    //método para añadir valores al ArrayList
    //el método recibe como parámetro el ArrayList de String
    //donde vamos a guardar los nombres introducidos por teclado
    public static void leer(ArrayList<String> A) {
        Scanner sc = new Scanner(System.in);
        String nombre, respuesta;
        do {
           System.out.print("Introduce un nombre: ");
           nombre = sc.nextLine();
           A.add(nombre);
           do {
               System.out.print("Continuar leyendo? (SI/NO)");
               respuesta = sc.nextLine();
           } while (!respuesta.equalsIgnoreCase("SI") && !respuesta.equalsIgnoreCase("NO"));
        } while (respuesta.equalsIgnoreCase("SI"));
    }

    //método para mostrar el ArrayList completo por pantalla
    //un elemento por línea
    public static void mostrar(ArrayList<String> A) {
       for(String s: A){
           System.out.println(s);
       }
        System.out.println();
    }

    //método que recibe un ArrayList de String y devuelve otro ArrayList de String
    //con el contenido invertido
    public static ArrayList<String> invertir(ArrayList<String> A) {
        // Crea un ArrayList para devolver el resultado del método
        ArrayList<String> B = new ArrayList<>();
        // Recorre el ArrayList A en orden inverso
        for (int i = A.size() - 1; i >= 0; i--) {
             // Añade cada String de A a B
             B.add(A.get(i));
        }
        return B;
    }
}
```

8. ARRAYS BIDIMENSIONALES UTILIZANDO ARRAYLIST

Un ArrayList es un array unidimensional, pero con ellos podemos *simular* arrays de dos o más dimensiones **anidando ArrayLists**.

Para crear un ArrayList bidimensional lo que creamos es un ArrayList cuyos elementos son a su vez ArrayList. Esto se puede extender sucesivamente y obtener ArrayList de más dimensiones.

Ejemplo:

Programa que lee las notas de 10 alumnos y las guarda en un ArrayList Bidimensional. Cada alumno tiene un número indeterminado de notas. La lectura de notas de cada alumno acaba cuando se introduce un número negativo. Finalmente se muestran todas las notas de todos los alumnos.

```
import java.util.ArrayList;

public class ArrayList5 {

  public static void main(String args[]){

        Scanner sc = new Scanner(System.in);
        final int numAlumnos = 3;  //número de alumnos
        int i, j, nota, cont = 1;

        //crear un ArrayList bidimensional de enteros vacío
        //Realmente se crea un ArrayList de ArrayLists de enteros
        ArrayList<ArrayList<Integer>> array = new ArrayList<>();

        //Se leen las notas de cada alumno.
        System.out.println("Introduzca notas. <0 para acabar");
        for(i = 0; i < numAlumnos; i++){
            cont = 1;
            System.out.println("Alumno " + (i+1) + ": ");
            System.out.print("Nota " + cont + ": ");
            nota = sc.nextInt();

            //para cada alumno se añade una nueva fila vacía
            //esto es necesario porque el arrayList se crea vacío
            array.add(new ArrayList<Integer>());

            while(nota>=0){
                array.get(i).add(nota); //en la fila i se añade un nueva nota
                cont++;
                System.out.print("Nota " + cont + ": ");
                nota = sc.nextInt();
            }
        }

        //Mostrar todas las notas
        System.out.println("Notas de alumnos");
        for(i=0;i<array.size();i++){  //para cada alumno (para cada fila)
            System.out.print("Alumno " + i + ": ");
            for(j=0;j<array.get(i).size();j++){ //se recorre todas la columnas de la fila
                System.out.print(array.get(i).get(j) + " "); //se obtiene el elemento i,j
            }
            System.out.println();
        }
  }
}
```