



## CICLO 2

[FORMACIÓN POR CICLOS]


# La clase **ARRAYLIST**



Ingeni@  
Soluciones TIC



UNIVERSIDAD  
DE ANTIOQUIA  
Facultad de Ingeniería



Como hemos visto, cada lenguaje de programación cuenta con sus propias formas de agrupar elementos. Esto, permite la gestión rutinaria de información en una aplicación, la cual es indispensable en casi todos los casos de uso de un lenguaje.

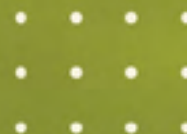
La forma más simple de agrupar datos en Java es hacerlo mediante un *array*, arreglo o vector. Un array nos permite definir un objeto que agrupa un número fijo de elementos del mismo tipo, para su posterior inicialización y manipulación. Sin embargo, los arrays son de tamaño fijo. Es decir, una vez los creamos, y definimos el número de sus elementos, no podemos aumentar o disminuir dicho tamaño. Por lo cual, si requerimos un array más grande, debemos crear otro con el número de elementos requerido, y así sucesivamente. Es por esto, que Java ofrece las colecciones.

En Java, una **colección** es un objeto que agrupa varios elementos del mismo tipo en una sola unidad. Las colecciones se suelen usar para almacenar, obtener, manipular y comunicar datos agrupados. Por lo anterior, son ampliamente usadas en todos los lenguajes de programación, si bien cada lenguaje tiene su propio conjunto de colecciones. Por lo general, dentro de una aplicación en desarrollo, una colección representa un grupo natural de elementos. Por ejemplo, un mazo de cartas, un grupo de alumnos, las áreas de una empresa, los empleados de una organización, entre muchos otros.

Dado que los grupos anteriores pueden funcionar de maneras diferentes y tener requisitos distintos dentro de una aplicación, Java ofrece una gran cantidad de clases, interfaces y algoritmos para la gestión de diferentes tipos de colecciones. Cada interfaz y cada clase funciona de forma diferente y tiene propósitos distintos, pero todas nos permitirán gestionar conjuntos de objetos. Como podrán adivinar, el mundo de las colecciones en Java es muy amplio, por lo que se recomienda explorar la documentación del lenguaje si se quiere profundizar al respecto<sup>1</sup>. Por nuestra parte, exploraremos la clase más usada de todas aquellas entre las colecciones en Java, la clase ArrayList.

---

<sup>1</sup>Documentación oficial sobre las colecciones en Java: <https://docs.oracle.com/javase/tutorial/collections/index.html>



Un **ArrayList** es un objeto de la clase del mismo nombre, con un desempeño aceptable en la mayoría de los casos y que funciona como un *array* (o arreglo) expandible de forma dinámica. Es decir, un **ArrayList** se comporta como un arreglo al cual le podremos agregar, modificar y quitar elementos de acuerdo a nuestras necesidades. Algo a tener presente es que los elementos de un ArrayList deben ser del mismo tipo, de manera similar a como ocurre con los arrays convencionales. Pero a diferencia de estos, no admite elementos de tipos de datos primitivos (int, float, char, etc.) en lugar de los cuales requiere clases de envoltura de tipos (o también llamadas *wrapper classes*).


Una clase de envoltura o **wrapper class** es una clase que se puede usar en lugar de un tipo de datos primitivo y cuyos objetos (variables) funcionan de manera similar a las variables normales, más específicamente, de manera similar a las de tipo String. Cada tipo de datos primitivo tiene una *wrapper class* equivalente y en todos los casos el valor por defecto será null, tal como se observa a continuación.

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

En resumen, un objeto de la clase ArrayList nos permitirá almacenar en un solo lugar un número variable de elementos del mismo tipo. El tipo podrá ser una clase o una interfaz, y en lugar de los tipos de datos primitivos, deberán ser elementos del tipo de la *wrapper class* equivalente.

Para usar una ArrayList, lo primero que debemos hacer es declararlo e instanciarlo, casi como cualquier objeto.

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



Arriba podemos observar que se crea un ArrayList de enteros llamado números. Tanto en la declaración como en la instanciación (las cuales estamos haciendo en una sola línea) indicamos entre símbolos de “mayor qué” y “menor qué” el tipo de los elementos que contendrá el ArrayList. En este caso, estamos creando un ArrayList de elementos Integer, es decir, de elementos de tipo int. La siguiente forma también es válida (y más corta).

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

Una vez hemos declarado e instanciado el ArrayList, podemos agregar elementos a este, mediante el método add.

```
numeros.add(8);  
numeros.add(427);  
numeros.add(-75);
```

También podemos saber el tamaño del arreglo con el método size, que en el siguiente caso imprimirá 3.

```
System.out.println(numeros.size());
```

Además, podemos acceder a cualquier elemento del ArrayList. Una de las formas de lograrlo es mediante el método get, pasando como argumento el índice del elemento. El índice es la posición donde se encuentra el elemento dentro del ArrayList, empezando a contar desde cero (0), tal como en los arrays. Por ejemplo, para el siguiente caso, imprimirá 427, que es el valor del elemento que se encuentra en la posición 1.

```
System.out.println(numeros.get(1));
```

También, siempre que el ArrayList no se encuentre vacío, podremos eliminar elementos. En el siguiente caso, eliminaremos el tercer elemento del arreglo (usando el método remove), y luego averiguaremos de nuevo el tamaño del ArrayList, que debería de ser 2.

```
numeros.remove(2);  
System.out.println(numeros.size());
```





Por otro lado, podemos vaciar el ArrayList por completo, usando el método clear, después de lo cual, este pasará a tener un tamaño de cero (0) elementos<sup>2</sup>.

```
numeros.clear();  
System.out.println(numeros.size());
```

Los anteriores son solo algunos de los múltiples métodos con los que cuenta la clase ArrayList, por lo que recomendamos consultar la referencia del lenguaje para familiarizarse con otros métodos que se puedan necesitar<sup>3</sup>.

A continuación, podremos comprobar que un ArrayList funciona de manera muy similar con objetos de clases que no corresponden a wrapper clases (tipos de datos primitivos)<sup>4</sup>.

<sup>2</sup>Del ejemplo completo se encuentra acá: [https://github.com/leonjaramillo/udea\\_ruta2\\_ciclo2/blob/main/main/java/co/edu/udea/udea\\_ruta2\\_ciclo2/PruebaArrayList.java](https://github.com/leonjaramillo/udea_ruta2_ciclo2/blob/main/main/java/co/edu/udea/udea_ruta2_ciclo2/PruebaArrayList.java)

<sup>3</sup>Referencia del lenguaje para ArrayList: <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

```

package co.edu.udea.udea_ruta2_ciclo2;

import co.edu.udea.udea_ruta2_ciclo2.poo.Estudiante;
import java.util.ArrayList;

public class PruebaOtroArrayList {
    public static void main(String[] args) {
        //Se declara e instancia un ArrayList de estudiantes
        ArrayList<Estudiante> inscritos = new ArrayList<>();
        //Se declaran e instancian los estudiantes de la clase
        existente
        Estudiante primero = new Estudiante("Juan", "Pérez",
16, 4.5);
        Estudiante segundo = new Estudiante("Pedro",
"Cañas", 19, 3.5);
        Estudiante tercero = new Estudiante("Lucas", "Ruiz",
18, 3.9);
        Estudiante cuarto = new Estudiante("Tomás", "Álvarez",
21, 4.1);
        //Se agregan los estudiantes al ArrayList
        inscritos.add(primeros);
        inscritos.add(segundo);
        inscritos.add(tercero);
        inscritos.add(cuarto);
        //Se verifica el tamaño del ArrayList
        System.out.println(inscritos.size());
        //Se elimina un elemento (el segundo)
        inscritos.remove(1);
        //Obtenemos el nombre del primer inscrito
        System.out.println(inscritos.get(0).getNombres());
        //Obtenemos el promedio del tercer inscrito
        System.out.println(inscritos.get(2).getPromedio());
    }
}

```


Lo anterior debería tener la siguiente salida.

```

4
Juan
4.1

```

<sup>4</sup>El ejemplo completo se encuentra acá: [https://github.com/leonjaramillo/udea\\_ruta2\\_ciclo2/blob/main/main/java/co/edu/udea/udea\\_ruta2\\_ciclo2/PruebaOtroArrayList.java](https://github.com/leonjaramillo/udea_ruta2_ciclo2/blob/main/main/java/co/edu/udea/udea_ruta2_ciclo2/PruebaOtroArrayList.java)



Si nos fijamos bien, en la clase anterior se puede apreciar el uso de la cláusula `import`. Las cláusulas **import** se sitúan, de ser necesarias, siempre después de la instrucción `package`, y con ellas debemos importar las clases que no se sitúan dentro del mismo paquete de la clase que estamos escribiendo.

En el caso anterior, las clases `ArrayList` y `Estudiante` se encuentran en paquetes diferentes a aquel donde está la clase `PruebaOtroArrayList`. Es por esto, que para cada una de esas clases se usa la cláusula `import`, seguida del nombre completo de la clase, anteponiendo el paquete en el que se encuentra y separándolos con un punto (`.`), así.

```
import co.edu.udea.udea_ruta2_ciclo2.poo.Estudiante;  
import java.util.ArrayList;
```

