



---

# PROGRAMACIÓN LINEAL CON JAVA

---

Programación



18 DE FEBRERO DE 2025  
ERIK AMO TOQUERO

## Contenido

Estructuras de Control Básicas de Java: .....	2
Entrada y salida de la información: .....	2
Estructuras selectivas: .....	3
Estructuras de control Iterativas: .....	5
Estructuras de Salto:.....	6
Funciones .....	7
Funciones sin devolución (vacíos, void) .....	7
Funciones con devolución .....	8
Funciones con uso de parámetros (esto sirve también para POO) .....	8
Funciones recursivas: .....	8
Arrays .....	9
Declaración en java: .....	9
Asignación y acceso.....	9
Comportamiento:.....	9
Paso de Valor: .....	10
Iniciación Rápida de un array: .....	10
Muestra de contenido de un Array: .....	10
Comparación de un Array: Arrays.equals(array1, array2) .....	10
Copia de arrays:.....	11
Eliminación de espacios en el Array: .....	11
Arrays Multidimensionales: .....	12
Recorrido de un array bidimensional por los ejes con ejemplo para meter los numeros: .....	12
Ver un array bidimensional entero:.....	12
Colecciones .....	13
ArrayList:.....	13
Diccionarios o mapas: .....	16



## Estructuras de Control Básicas de Java:

### Entrada y salida de la información:

El flujo estándar de la salida de información es `System.out`, y, según qué se quiera mostrar y cómo se quiera hacer, hay diferentes métodos.

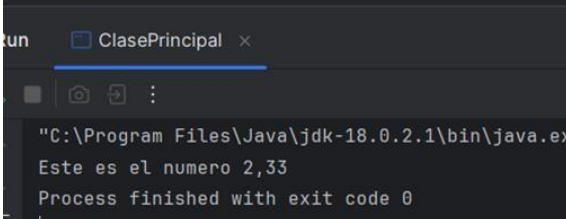
Los estudiados en clase para sacar información por pantalla son los PRINT (literalmente imprimir):

- **`System.out.print(String)`:** Saca por pantalla literalmente lo que se ponga en el string. No añade saltos de línea ni nada
- **`System.out.println(String)`:** Saca por pantalla el texto que haya en el string, y añade un salto de línea al acabar.
- **`System.out.printf(String con formato, valores)`:** Este puede resultar más lioso, ya que a primera vista funciona igual que un print normal, pero según las opciones de formato que demos en el string, funcionará de una forma u otra:

- o Formato decimal: `%.f`

Sintaxis: `%.xf`, siendo x el número de decimales que queramos que tenga el número.

```
double num = 2.3333333;  
System.out.printf("Este es el numero %.2f", num);
```



run ClasePrincipal x

"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe"  
Este es el numero 2,33  
Process finished with exit code 0

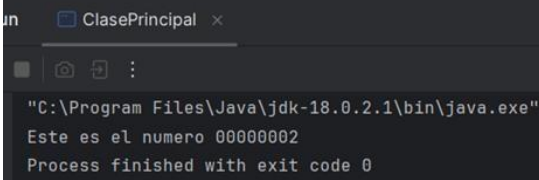
- o Relleno a la izquierda: `%d`

Sintaxis: `%Axd`, siendo A el carácter numérico que queramos poner y x la cantidad de cifras que buscamos que tenga el número. Ejemplo:

`"%05d"` con 23

saldrá "00023"

```
int num=2;  
System.out.printf("Este es el numero %08d", num);
```



run ClasePrincipal x

"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe"  
Este es el numero 00000002  
Process finished with exit code 0

- o Booleano: `%b`

Sintaxis: `%b`, para sacar true o false con una booleana en un formato

```
boolean algo = true;
System.out.printf("El booleano esta en %b", algo);
```

Run ClasePrincipal x

```

C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe
El booleano esta en true
Process finished with exit code 0

```

- Representación en hexadecimal (%x) y en octal (%o)

```
int num = 18;
System.out.printf("18 en octal es %o, y en hexadecimal es %x", num, num);
```

Run ClasePrincipal x

```

C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program
18 en octal es 22, y en hexadecimal es 12
Process finished with exit code 0

```

- Hora actual: %tT
- Simbolo de porcentaje en formato: %%

```
public static void main(String[] args) {
    System.out.printf("El porcentaje de 1 entre 2 es 50%%");
}
```

Run ClasePrincipal x

```

C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-j
El porcentaje de 1 entre 2 es 50%
Process finished with exit code 0

```

## Estructuras selectivas:

Entramos en el campo de las selecciones, donde se empieza a tener en cuenta el valor de “verdadero” y “falso” que nos dan las operaciones booleanas

### IF(condición):

Es la estructura más sencilla. Lo que hace es que, si la condición se cumple, siga con lo que hay dentro de su rama. De no ser así, simplemente pasa de ello.

```
Scanner scan = new Scanner(System.in);
System.out.println("Si me dices el numero 3 te digo que es tres");
int num = scan.nextInt();
if(num==3){
    System.out.println("Es tres");
}
```

Run ClasePrincipal x

```

C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Pr
Si me dices el numero 3 te digo que es tres
3
Es tres

```

```
Scanner scan = new Scanner(System.in);
System.out.println("Si me dices el numero 3 te digo que es tres");
int num = scan.nextInt();
if(num==3){
    System.out.println("Es tres");
}
```

Run ClasePrincipal x

```

C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Pr
Si me dices el numero 3 te digo que es tres
4
Process finished with exit code 0

```

## IF(condición) - else:

Else es un complemento de "if" para que, en caso de que la condición marcada no se cumpla, el código haga lo que dicte "else". Permite encadenar ifs yelses seguidos

```
System.out.println("Si me dices el número 3 te digo que es tres");
int num = scan.nextInt();
if(num==3){
    System.out.println("Es tres");
}else{
    System.out.println("No es tres");
}

Run ClasePrincipal x
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe"
Si me dices el numero 3 te digo que es tres
3
Es tres
```

## Operador Ternario (if-else resumido):

A la hora de usar una estructura if else para dar un valor a una variable, se puede hacer más fácil con un operador ternario.

Tienen esta estructura: variable=(condición)? Verdadero : Falso

Si la condición es verdadera, la variable toma el valor de la izquierda, y si es falso, el de la derecha.

```
int num, mayor=12;
System.out.println("Escribe un número, yo diré otro y te diré cuál es mayor. Yo digo el 12");
num=scan.nextInt();
mayor=(mayor<num)? num : mayor;
System.out.println("El mayor es " + mayor);

Run ClasePrincipal x
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe"
Escribe un número, yo diré otro y te diré cuál es mayor. Yo digo el 12
12
El mayor es 12
Process finished with exit code 0
```

## Switch(variable):

Para acortar a veces una cadena de ifs y elses repetitivos, se utiliza el comando switch. Este se encarga de buscar entre varias condiciones a la vez a ver cuál se cumple. Dependiendo de cual sea esa condición, hará diferentes cosas.

Puedes poner cuantos **casos** quieras, siempre teniendo en cuenta que, si quieres poner algún valor por defecto tienes que hacer un caso llamado **default**.

En el switch es muy importante poner **break**, un comando de salida de estructura

```
System.out.println("Dime un número del 1 al 5");
int num = scan.nextInt();
switch (num){
    case 1:
        System.out.println("Es 1");
        break;
    case 2:
        System.out.println("Es 2");
        break;
    case 3:
        System.out.println("Es 3");
        break;
    case 4:
        System.out.println("Es 4");
        break;
    case 5:
        System.out.println("Es 5");
        break;
    default:
        System.out.println("No esta entre el 1 y el 5");
}
```

## Estructuras de control Iterativas:

Las también estructuras iterativas, también llamadas “bucles”, son estas que repiten una serie de comandos dentro de una rama las veces que nos interese. De ahí podemos hacer bucles infinitos con el objetivo de finalizar una vez se cumple una condición (o no).

### While(condición):

Este bucle es uno de los más utilizados. Se podría leer como “Mientras se cumpla esto se hace esto”, es decir, mientras la condición que marques sea verdadera, el bucle se seguirá repitiendo.

```
Scanner scan = new Scanner(System.in);
int num=0;
while(num>=0){
    System.out.println("Dime números positivos");
    num=scan.nextInt();
}
System.out.println("Ese era negativo");
```

Run ClasePrincipal x

```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe"
Dime números positivos
3
Dime números positivos
4
Dime números positivos
-1
Ese era negativo
Process finished with exit code 0
```

### Do-While(condición):

Su funcionamiento es prácticamente el mismo, pero con un ligero cambio.

Con “While” desde el principio tiene que ser verdadero el valor, mientras que con do-while, te deja meterte en el bucle una primera vez antes de comprobar si se cumple la condición.

En el siguiente ejemplo se va a ver que no funciona el while y sí funciona el do-while:

<pre>int num=0; while(num&gt;0){     System.out.println("Dime números mayores que 0");     num=scan.nextInt(); } System.out.println("Ese es 0 o menor que 0");</pre>	<pre>Scanner scan = new Scanner(System.in); int num=0; do {     System.out.println("Dime números mayores que 0");     num=scan.nextInt(); } while(num&gt;0); System.out.println("Ese es 0 o menor que 0");</pre>
<p>ClasePrincipal x</p> <pre>"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-jav Dime números mayores que 0 Ese es 0 o menor que 0 Process finished with exit code 0</pre>	<p>ClasePrincipal x</p> <pre>"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-jav Dime números mayores que 0 Dime números mayores que 0 0 Ese es 0 o menor que 0 Process finished with exit code 0</pre>

### For(contador, condición, aumento/decrecimiento):

La estructura for se divide en tres partes:

- 1: Declaración de una variable (puede estar iniciada fuera). Lo que haremos con esto es hacer una variable que funcione como un contador, o sea, que aumente o disminuya según nos interese. Puede estar iniciado en 0 o cualquier otro número que queramos.
- 2: Condición: el bucle continúe y el contador aumenta mientras esa condición sea verdadera, por eso a menudo se hace con el contador, aunque no es del todo necesario.
- 3: Aumento/Decrecimiento: Esto va a hacer que nuestro contador suba o baje como nos interese. Lo más sencillo para hacer es que sume o reste, dándole el paso que nos interese. Por ejemplo, para que sume con paso uno es ponerle `i++` o `i--`—si queremos ir restando, pero se puede poner de 10 en 10 (`i=i+10`), duplicándose (`i=i*2`) y demás.

```
int num=0;
for(int i=0;i<5;i++) {
    System.out.println("i es " + i);
    num = num + i;
    System.out.println("Se ha sumado así que num ahora es " + num);
}
```

```
ClasePrincipal
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Progr
i es 0
Se ha sumado así que num ahora es 0
i es 1
Se ha sumado así que num ahora es 1
i es 2
Se ha sumado así que num ahora es 3
i es 3
Se ha sumado así que num ahora es 6
i es 4
Se ha sumado así que num ahora es 10
```

### Estructuras de Salto:

Como indica su nombre, estas estructuras sirven para salir de un bucle o de alguna estructura como if, switch y demás. Se distinguen 2 perfectamente, pero voy a mencionar una tercera que no deja de ser un salto.

#### Break:

Esta se utiliza en los switch para evitar que haya saltos entre casos de forma innecesaria. Lo que hace el break es que, cuando llega a esta línea, abandona el bucle o la estructura tal y como esté. Se puede utilizar también en caso de hacer algún bucle con condiciones o directamente infinitos, como es el ejemplo a continuación:

En este ejemplo es una mini estupidez que se podría arreglar con un `while(num!=5)`, sí, pero es para ejemplificar.

```
while(true){
    System.out.println("Escribe 5");
    num=scan.nextInt();
    if(num==5){
        break;
    }
}
System.out.println("Gracias");
```

```
ClasePrincipal
"C:\Program Files\Java\jdk-18.0.2.1\bin\j
Escribe 5
2
Escribe 5
3
Escribe 5
5
Gracias
Process finished with exit code 0
```

### Continue:

Continue sirve para que, en bucles, bajo ciertas condiciones, se omita parte del comando que haya en este alguna que otra vez. Por ejemplo: Ponemos un for que nos diga los números del 1 al 5 pero que se salte el 4:

Todo lo que haya después del continue y antes del cierre del bucle lo omitirá el código en esa condición.

```
for(int i=1; i<=5; i++){
    if(i==4){
        continue;
    }
    System.out.println(i);
}
```

ClasePrincipal x

"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe"

1  
2  
3  
5

Process finished with exit code 0

### System.exit(código de salida)

Este no es tal como un comando de salida de un bucle, sino que sirve para salir directamente del código. Dentro tienes que meter algún número para indicar un código que diga el por qué se ha cerrado, el 0 universalmente es conocido como que el programa ha finalizado tal y como debería, por ejemplo.

```
public static void main(String[] args) {
    for(int i=1; i<=5; i++){
        if(i==4){
            System.exit( status: 0);
        }
        System.out.println(i);
    }
}
```

ClasePrincipal x

"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe"

1  
2  
3

Process finished with exit code 0

La diferencia que se puede ver con el continue en este ejemplo es que se ha cerrado y no ha seguido con el 5, pero de haber más código después del for, tampoco lo haría, ya que es como si le diéramos a cerrar el programa.

## Funciones

Las funciones (más adelante llamados métodos) son fragmentos de código a parte del principal, hechos para simplificar la lectura y reducir la repetición del texto. Hay varios tipos de funciones:

### Funciones sin devolución (vacíos, void)

Son aquellas funciones que no devuelven valores, se declaran como [visibilidad] [static] void nombre(parámetros){}. La llamada es simplemente escribir el nombre de la función y los valores a pasar

```
public void func1(){
}
public void main(){
    func1();
}
```



## Funciones con devolución

Son las funciones que devuelven algún valor, ya sea int, boolean, Object...

Deben llevar el comando **return**, donde devolverá el valor elegido, y omitirá todo lo que haya a continuación.

Luego, en la llamada de la función, necesitan ser utilizados para algo, es decir, se deben almacenar o utilizar en otro comando.

```
public void main(){
int funcion = func1();
} public int
func1(){      return
1;
}
```

## Funciones con uso de parámetros (esto sirve también para POO)

Las funciones pueden tener parámetros en su composición, es decir, variables que les pasamos nosotros desde otro lugar y que pueden utilizar. En la creación de la función, se hace:

[visibilidad] [static] {int/boolean/void/...} nombre (Tipo valor1, Tipo valor2,...){}

Luego, en la llamada a la función tienes que pasar los valores en el mismo orden que en la creación, **sin importar los nombres que tengan cada variable**

```
public void main(){
int a = 10;      String
b = "Hola";
func1(b, a);
}
public void func1(String nombre, int numero){
System.out.println(nombre + ", " + numero); }
```

## Funciones recursivas:

Son utilizadas para las matemáticas más que otra cosa. Lo que hace es llamar a la función desde dentro de la propia función, haciendo un bucle que acaba cuando le interesa a alguien:

Un ejemplo es la sucesión de Fibonacci:

```
public static void main(String[] args) {
Scanner scan = new Scanner(System.in);
int pos;      do {
    System.out.println("Escribe una posición en la secuencia de Fibonacci y
diré su número(Empieza en 0)");
pos = scan.nextInt();
}while(pos<0);
int num=Fibonacci(pos);
System.out.println(num + " es la posición numero " + pos + " de la
secuencia de Fibonacci.");
}
public static int Fibonacci(int pos){
if(pos==1||pos==0){      return 1;
} else{
    return Fibonacci(pos-1)+Fibonacci(pos-2);
}
}
```

Para poner un ejemplo, voy a elegir la 4 posición:

- 4 es distinto de 1 y de 0, por lo que ahora hacemos la sucesión de 3 mas la sucesión de 2  
○ 2 no es ni 1 ni 0, así que se hace la sucesión con 0 y 1,  $1+1=2$ , sucesión de 2 = 2  
○ 3 no es ni 1 ni 0, así que se hace sucesión con 1 y 2, sucesión de 3 = sucesión de 2 + 1  
▪ 2 no es ni 1 ni 0, así que se hace la sucesión con 0 y 1,  $1+1=2$ , sucesión de 2 = 2  
○ Sucesión de 3 =  $1 + 2 = 3$
- Sucesión de 4 =  $3 + 2 = 5$ ;

**Solución: 5**

Comprobación: Fibonacci = 1, 2, 3, **5**, 8, 13, 21...

## Arrays

Los arrays son estructuras estáticas que representan un contenedor de datos del mismo tipo, tiene un tamaño fijo y puede almacenar todo tipo de datos

Si su tamaño es 4, recorrerá de 0 a 3 por ejemplo, es decir, los arrays empiezan en la posición 0

Los arrays son objetos, por lo que hay que declararlos e inicializar sus elementos

### Declaración en java:

tipo []nombre; o en su defecto tipo nombre[]; Ejemplo:

nombre = new tipo[longitud];

```
int []MyArray;  
MyArray = new int[3];
```

Formas alternativas de hacer arrays:

```
int []MyArray2 = new int[3]; int  
[]MyArray3 = {2,3,4};
```

### Asignación y acceso

Hay que llamar a una posición específica del array para introducir un valor:

```
MyArray[0]=2;
```

### Comportamiento:

Los arrays no funcionan igual que las demás variables. Si igualas dos arrays y cambias el valor de un array, se cambia el valor en el otro (Explicación: Los arrays llevan una dirección de memoria, eso lo que hace es que al igualar dos arrays igualas su dirección de memoria, por lo que se convierten en literalmente el mismo array)

```
int a=7; int  
b=a; b=9;  
Final: a=7, b=9  
  
MyArray2=MyArray3;  
MyArray2[1]=2;  
Final: MyArray2[1]=2, MyArray3[1]=2
```

### Paso de Valor:

Las funciones pueden cambiar valores de los arrays pasando estos directamente, es decir, podemos pasar un array como valor para una función. Ejemplo:

```
public static void main(String[] args) { int
    []MyArray4 = new int[4];
    System.out.println("Array en 0 original: " + MyArray4[0]); //0
    modificarArray(MyArray4);
    System.out.println("Array en 0 modificada: " + MyArray4[0]);
//10 }
public static void modificarArray(int[]array){
array[0]=10;
}
```

### Iniciación Rápida de un array:

Para iniciar un array, la forma más sencilla sería hacer un bucle for (o forEach) que recorra todo el array y vaya metiendo valores, pero hay formas más sencillas:

```
int []MyArray5= new int[5];
Arrays.fill(MyArray5, 5);
```

La función **Arrays.fill(Array, valor)** sirve para rellenar TODOS los huecos de un array con un valor

### Muestra de contenido de un Array:

Hay dos formas:

- **For/ForEach**

```
for(int num : MyArray5){
    System.out.println(num);
} //ForEach

for(int i = 0; i< MyArray5.Length; i++){
    System.out.println(i);
} //for de un array usando su longitud (.length)
```

- **Arrays.toString(Array):** Muestra de una forma predefinida todo el contenido del Array

```
System.out.println(Arrays.toString(MyArray));
```

### Comparación de un Array: Arrays.equals(array1, array2)

Hace falta la librería de Arrays, concretamente la función "equals", la cual devuelve un booleano. Si apuntan a la misma dirección (Es decir, son iguales), siempre va a ser verdadero.

```
int []array = {1,2,3,4}; int
[]array2 = {1,2,3,5};
System.out.println("¿Los arrays son iguales?" +
Arrays.equals(array,array2)); //Falso
```

## Copia de arrays:

La forma más "correcta" de hacerlo es con `Arrays.copyOf(array, longitud)`

```
int[] numeros = {1,2,5,3,5}; int[] numeros2 =  
Arrays.copyOf(numeros, numeros.Length);
```

Tiene otra variante, `Arrays.copyOfRange(array, posPrincipal, posFinal)`

```
int[] numeros = {1,2,5,3,5}; int[] numeros2 =  
Arrays.copyOfRange(numeros, 0, 2); //Copia las posiciones del 0  
al 2
```

Para acabar, de forma más completa, está `System.arraycopy(arrayOrigen, posicionOrigen, arrayDestino, posicionDestino, longitudCopia)`

```
int[] numeros = {1,2,5,3,5}; int[]  
numeros2 = new int[5];  
System.arraycopy(numeros, 2, numeros2, 3, 2); /*  
Final:
```

```
    numeros[1, 2, 5, 3, 5];  
    Numeros2[0, 0, 0, 5, 3];
```

Copia tantos numeros de la posicion origen del array 1 desde la posicion destino del array 2 como numeros digamos en la longitud  
\*/

Esta se usa mucho para modificar posiciones

## Eliminación de espacios en el Array:

Hay diferentes formas según lo que se pida:

1. Se nos pide eliminar el último espacio del array:

```
array = Arrays.copyOf(array, array.Length-1); //Copias el array sin el  
último espacio dentro del propio array que estás modificando
```

2. Se nos pide eliminar el espacio "num" del array:

```
int[] array = {1,2,3,4,5,6,7};  
System.arraycopy(array, num, array, num+1, array.Length-num);  
//pasas todo el array desde la posicion num+1 a la posición num  
(mueves todo una posición atrás para quitar el hueco de num) array =  
Arrays.copyOf(array, array.Length-1); //Quitas el hueco que te sobra
```

## Arrays Multidimensionales:

Los arrays múltiples son arrays que contienen otros dentro. (Array de arrays)

Se declaran como si fuera un array normal, pero con dos corchetes para hacer un contenedor

En el caso de arrays bidimensionales, en términos de una tabla, el primer corchete hace función de fila, y el segundo corchete de columna (truco para acordarse: FC de Fútbol Club) (gracias fifas)

```
int[][] arrayBidimensional = new int[2][3];
//Creación de un array bidimensional de dos filas y tres columnas.

int[][] arrayBidimensional2 = {{1,2,3}, {1,2,2}};
//Creación de un array bidimensional de dos filas y tres columnas con
datos directamente
```

Los arrays bidimensionales pueden tener distinto número de elementos si no se declara por defecto, y se puede rellenar fila a fila:

```
int[][] matriz = new int[3][];

matriz[0] = new int[]{1, 2, 3}; matriz[1] = new int[]{4,
5, 6, 7}; matriz[2] = new int[]{8, 9};
```

Recorrido de un array bidimensional por los ejes con ejemplo para meter los números:

```
Scanner sc = new Scanner(System.in); int[][]
numeros = new int[2][3]; for (int i = 0; i <
numeros.Length; i++) {    for (int j=0;
j<numeros[i].Length; j++){
    System.out.println("Escribe el numero a guardar");
    numeros[i][j] = sc.nextInt();    } }
```

### Ver un array bidimensional entero:

Se puede mirar por varios métodos:

- **Método deepToString(array)**

Se utiliza para ver todo el array de golpe. Mostrando el contenido de cada array por separado

```
System.out.println(Arrays.deepToString(matriz));
```

- **Método toString en for**

```
for (int i = 0; i < matriz.Length; i++) {
    System.out.println(Arrays.toString(matriz[i]));
} //Va haciendo un toString de cada array dentro del array principal
```

- **Método del doble for**

```
for (int i = 0; i < matriz.Length; i++) {    for (int j=0;
j<matriz[i].Length; j++){
    System.out.print(matriz[i][j] + " ");
}
System.out.println(); }
```

## Colecciones

Los arrays son estructuras estáticas, las cuales tienen ciertas dificultades a la hora de introducir datos nuevos o borrar/cambiar otros. Por ello, Java proporciona una serie de estructuras dinámicas que comparten un conjunto de métodos declarados en la interfaz Collection. Todas ellas implementan dicha interfaz, aunque de diferentes formas. Se conocen las Listas, Conjuntos matemáticos (set), y diccionarios o mapas.

### ArrayList:

Los ArrayList (lista de arrays) son objetos que almacenan contenido como si fueran arrays, pero siguen una estructura más “sencilla” que los arrays:

#### Declaración:

```
ArrayList<[Variable]> arrayList = new ArrayList<>();
```

En [Variable] habrá que poner el tipo de valor que va a almacenar este ArrayList (Integer, Double, Long, incluso Objetos)

#### Añadir contenido:

Hay dos formas de aplicar el mismo método:

- Indicando índice: **arrayList.add(indice, valor)**

```
ArrayList<Integer> arrayList = new ArrayList(); arrayList.add(3, 5); //Añade un 5 en el índice 3
```

- Sin indicar un índice (añade al final): **arrayList.add(valor)**

```
ArrayList<Integer> arrayList = new ArrayList(); arrayList.add(1); //Añade una posición y mete un 1
```

Cambio de un valor por medio del índice: **arrayList.set(indice, valornuevo)**:

```
Integer[] num = {1, 4, 6, 2, 6565};
ArrayList<Integer> arrayList = new ArrayList<>();
arrayList.addAll(Arrays.asList(num)); arrayList.set(2, 5); //En la posición 2 cambia el 6 por un 5
```

Sacar el valor de un índice: **arrayList.get(indice)**

```
arrayList.get(1); //Obtiene el valor del índice 1
```

Borrar un valor por su índice: **arrayList.remove(indice)**

```
arrayList.remove(9); //Borra el valor del índice 9
```

Borrar un valor por su propio valor: **arrayList.remove(valor)**

```
ArrayList<Integer> arrayList = new ArrayList<>();
arrayList.remove(Integer.valueOf(2)); //Borra los enteros de valor 2
```

Borrar todo el contenido de una: **arrayList.clear()**

```
arrayList.clear(); //Borra directamente todo lo que hay en el arrayList
arrayList.retainAll([Coleccion]); //Borra todo lo que no pertenezca a otra lista
arrayList.retainAll([Coleccion]); //Borra todo lo que pertenezca a otra lista
```

Búsqueda por Índice `arrayList.indexOf(valor)`:

```
arrayList.indexOf(Objeto); //Busca el índice al que pertenece x objeto
por primera vez (si hay varios iguales). Si no aparece, devuelve un -1
arrayList.lastIndexOf(Objeto); //Busca el índice al que pertenece x
objeto por última vez (si hay varios iguales). Si no aparece, devuelve
un -1
```

Booleana de comprobación:

- `arrayList.equals(Objeto)`: Da true si tienen los mismos elementos

```
ArrayList<Persona> arrayList = new ArrayList<>();
Persona paco = new Persona("123", "Nombre", true, true);
arrayList.add(paco);
boolean equals = arrayList.equals(paco);
```

- `arrayList.containsAll(Colección)`: Devuelve true si una contiene los elementos de otra colección

```
Integer[] num = {1, 4, 6, 2, 6565};
ArrayList<Integer> arrayList = new ArrayList<>();
arrayList.addAll(Arrays.asList(num)); arrayList.add(2); boolean b =
arrayList.containsAll(Arrays.asList(num)); //Es true, porque
contiene la lista num.
```

Función de paso de Lista a Array `arrayList.toArray()`:

Pasa el contenido de un ArrayList (lista dinámica) a un Array (conjunto estático)

```
ArrayList<String> arrayList = new ArrayList<>();
arrayList.add("a");
String[] listArray = (String[]) arrayList.toArray();
```

Función de paso de Array a Lista `Arrays.asList()`:

Pasa el contenido de un Array a formato de Conjunto para poder meterlo en una lista con el comando `addAll`

```
Integer[] num = {1, 4, 6, 2, 6565};
ArrayList<Integer> arrayList = new ArrayList<>();
arrayList.addAll(Arrays.asList(num));
```

Recorrido del ArrayList:

Hay 3 formas:

1. Con un bucle For

```
ArrayList<Integer> arrayList = new ArrayList<>(); for
(int i = 0; i<arrayList.size();i++){
    System.out.println(arrayList.get(i));
}
```

2. Con un bucle ForEach

```
ArrayList<Integer> arrayList = new ArrayList<>();
for (Integer numero : arrayList){
    System.out.println(numero);
}
```

3. Con un Iterador



Explicación rápida: Los iteradores son valores que generan las listas, y lo único que tenemos que hacer es llamarlos. Sirven para recorrer las listas, entre otros.

```
ArrayList<Integer> arrayList = new ArrayList<>();
Iterator<Integer> iterator = arrayList.iterator(); while
(iterator.hasNext()){
    Integer numero = iterator.next();
    System.out.println(numero + "\n"); }
```

Eliminación de contenido con un Iterator: iterator.remove:

```
ArrayList<Integer> arrayList = new ArrayList<>();
Iterator<Integer> iterator = arrayList.iterator(); while
(iterator.hasNext()){
    Integer numero = iterator.next();
    if (numero < 10){
        iterator.remove();
    }
}
```

Ejercicio con uso de listas y sets

```
public class Listas {
    public static void main(String[] args) {
        List<Integer> lista = new ArrayList<>();
        for (int i=0; i<20; i++){
            lista.add((int) (Math.random()*10)+1);
        }
        lista.sort(null);
        System.out.println("Lista original " + lista);

        Set<Integer> listaSinRepeticiones = new TreeSet<>(); //Creo un set que me
quita los repetidos listaSinRepeticiones.addAll(lista); //Añado toda una
lista
        System.out.println("Lista sin repeticiones " + listaSinRepeticiones);

        for (int i : listaSinRepeticiones){
            lista.remove(Integer.valueOf(i)); //Elimino la primera ocurrencia que
tenga el valor de i
        }

        Set<Integer> listaSoloRepetidos = new TreeSet<>();
        listaSoloRepetidos.addAll(lista);
        System.out.println("Lista solo con los repetidos: " + listaSoloRepetidos);

        Set<Integer> listaSoloUnaVez = new TreeSet<>();
        listaSoloUnaVez.addAll(listaSinRepeticiones); //Añado toda una colección
listaSoloUnaVez.removeAll(listaSoloRepetidos); //Elimino todos los elementos que
aparezcan en esta otra

        System.out.println("Lista con elementos que aparecen solo una vez:" +
listaSoloUnaVez);
    }
}
```



## Diccionarios o mapas:

Los diccionarios son estructuras dinámicas cuyos elementos son pares clave/valor en vez de un solo valor. Los diccionarios no heredan Collection y cada elemento tiene una clave única.

Los mapas se dividen en dos partes: una clave y un valor:

- La clave es el identificador, lo que, como bien indica el nombre, identifica esa posición exacta del mapa
- El valor es la parte del mapa que se guarda, se accede a ella mediante el identificador que tiene asignado

Se identifican tres tipos distintos:

- HashMap: No garantiza un orden particular
- TreeMap: Garantiza el orden basándose en el orden de los elementos insertados -  
LinkedHashMap: Inserta al final

### Declaración:

```
Map<k, v> m = new HashMap<>();
```

Donde k es la clave y V el otro valor

```
Map<Integer, String> mapa = new HashMap<>();
```

### Introducción de datos:

La introducción de datos se hace con el comando "put" y dos valores. Hay que tener en cuenta que la clave solo puede aparecer **una** vez, por lo que dará un error si se añade una clave por segunda vez

```
mapa.put(1, "Hola");
```

### Acceso de datos en un mapa:

Con el comando get(k), siendo K el identificador, se puede obtener el valor del mapa

```
mapa.get(1);
```

### Comprobador de valor y clave (booleana)

Esta booleana comprueba si en el mapa se encuentra el valor o la clave que se pide

```
Boolean boolKey = mapa.containsKey(5); //Busca la clave  
Boolean boolValue = mapa.containsValue("Adios"); //Busca el valor
```

### Entradas del mapa (Map.Entry)

Esto sirve para poder recorrer todo el mapa, y poder cambiar valores u obtener la clave y el valor.

```
for(Map.Entry<Integer, String> elemento : mapa.entrySet()){ }
```

Hay que tener en cuenta antes de hacer nada, que va a recorrer TODO el mapa.

Con esto se pueden hacer cosas tales como:

*Obtener un valor:*

```
String nombre = elemento.getValue();
```

*Obtener una clave:*

```
int clave = elemento.getKey();
```

*Cambiar el valor de una clave:*

```
elemento.setValue("Buenas");
```

Extras:

*Se puede guardar la lista de valores en una colección a parte:*

```
Collection<String> valoresMapa = mapa.values();
```

*Se puede guardar la lista de claves en un set:*

```
Set<Integer> clavesMapa = mapa.keySet();
```