

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM					
	MÓDULO	Programación						CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:				
	AUTOR	Francisco Bellas Aláez (Curro)							

Tema 5 – Arrays, Colecciones y Cadenas.

Inicialmente la forma en que almacenábamos datos en Java era muy simple: en cada variable un dato. Posteriormente en cuanto empezamos a aprender algo de POO vimos la manera de meter en un único objeto varios datos: sus campos o propiedades. Pero aún así estos métodos no son demasiado cómodos si lo que se necesita es almacenar una cantidad de datos muy grande.

Supongamos por un momento que necesitamos almacenar la temperatura media de 4 ciudades en un mes. Nos llegarían 4 variables y es manejable. Pero si necesitamos ampliarlo para que me acepte 100 ciudades sería mucho más complejo (¡100 variables!). Y si por encima quisiera tener la información de esas ciudades a lo largo de los doce meses del año el problema se acrecentaría notablemente.

Para solucionar esto veremos en este tema distintas estructuras de datos que nos facilitan el trabajo con grandes cantidades de información. Estas estructuras serán por un lado los *arrays* que son tablas de datos de tamaño fijo y las colecciones cuyo tamaño puede ir creciendo o decreciendo según las necesidades del programa.

Mediante estas estructuras podemos tanto gestionar grandes cantidades de datos como de mantener estructuras que representan ciertos elementos del mundo real como un tablero de juego o un entorno virtual.



<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Arrays unidimensionales

Un **array** es un conjunto de tamaño determinado de valores indexados que tienen todos el mismo tipo. El hecho que sean unidimensionales significa que es una lista simple de datos a los cuales se accede mediante un único índice.

Ejemplo: Tabla de temperaturas de 4, 6, 1000 o más ciudades. Gracias a un array puedo acceder mediante una única variable y un índice a cada una de las ciudades.

Nombre del array: temperaturas.

Elementos del array:

temperaturas[0] = 7.5

temperaturas[1] = 3.2

...

Tipo de dato de cada elemento: nº real (double ó float)

Índice del array: 0,1,2,....

En el ejemplo anterior vemos las partes del array:

- Por un lado el **nombre** que representa a toda la tabla de datos.
- Cada uno de los **datos**, que son todos **del mismo tipo**.
- El **índice** de cada dato mediante el cual accedemos al mismo.

En muchas ocasiones a los **arrays unidimensionales** se les denomina **vectores** (no confundir con el concepto de vector en matemáticas o física).

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Declaración y uso básico

En Java la declaración del objeto array se realiza mediante el tipo base y corchetes y se establece simplemente el nombre. Dichos corchetes también pueden ponerse a continuación del nombre si se quieren declarar otros enteros en dicha línea. Por ejemplo:

```
double[] temperaturas, valores; //Dos arrays
int edades[], numero; //Un array y un entero
```

En cualquier caso debe recordarse que la **buena práctica es declarar una variable por línea**.

Como un array **es un objeto**, para crearlo es necesario instanciarlo mediante **new**. Es precisamente al instanciarlo cuando se indica el tamaño del mismo de la siguiente forma:

```
temperaturas=new double[4];
```

en este caso creamos un array denominado **temperaturas** que puede contener 4 valores de tipo **double**. Fíjate que la sintaxis cambia ligeramente a lo que vimos en objetos ya que el constructor que va junto al new es ligeramente distinto.

Los índices son 0, 1, 2 y 3 de forma que puedo realizar operaciones como:

```
temperaturas[0]=7.5;
temperaturas[1]=3.2;
System.out.println(temperaturas[0]);
```

La indexación en java **siempre empieza en 0**.

También puedo aprovechar un bucle para rellenar, mostrar o realizar cualquier otra operación sobre el array. De hecho ahí está el interés de la gestión con arrays.

```
for (int i =0; i<temperaturas.length; i++){ // Rellena un array
    temperaturas[i]=Math.random()*50-25;
}
```

Para obtener el tamaño del array debo usar la propiedad **length** del objeto.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

```
for (int i =0; i<temperaturas.length; i++){ // Muestra un array
    System.out.printf("Indice:%2d    valor:%7.2f°C\n", i,
                      temperaturas[i]);
}
```

Ojo, si tratamos de mostrar un array sin bucle nos mostrará su dirección de memoria como cualquier otro objeto.

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

Existe una **excepción** que es cuando el **array es de tipo char**, en ese caso sí funcionaría directamente un print.

Aquí es donde estriba la potencia de los arrays: puedo acceder a gran cantidad de datos simplemente a través de índices.

Hay que tener dos cosas en cuenta sobre los **índices** en Java:

- Siempre **empieza en 0**: Con esto hay que tener cuidado porque conlleva a que el límite superior es la cantidad de elementos menos uno (**length-1**). Es un error muy típico salirse de este rango lo que provoca un fallo de programa en tiempo de ejecución. Pruébalo accediendo a la posición length.
- El tipo del índice si se gestiona con una variable debe ser un **tipo compatible con int** salvo long. Es decir: int, byte, short o char (**long NO**).



COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Si se quiere crear un array con unos valores predefinidos se puede hacer de la siguiente forma:

```
int v[] = { 23, 34, 1, -3, 6, 9 };
```

Este método es solo válido si se declara y asigna en la misma sentencia. Si necesitáramos hacerlo por separado, se debe usar new de la siguiente forma:

```
int v[];
v = new int[] { 23, 34, 1, -3, 6, 9 };
```

Arrays como parámetros

Un array puede ser pasado como parámetro al igual que cualquier otro objeto. Lo único que se debe tener en cuenta es que por ser una referencia a un objeto, si dentro de la función se modifica el array este queda modificado. Prueba el siguiente programa:

```
public static void cambio(double[] vector){
    vector[3] = 4.5;
}

public static void mostrarVector(double[] vector){
    for (int i=0; i < vector.length; i++) {
        System.out.printf("%5.1f", vector[i]);
    }
    System.out.println();
}

public static void main(String[] args) {
    double[] temperaturas = {7.5, 3.2, 1.1, 5.0, 10.3};

    mostrarVector(temperaturas);
    cambio(temperaturas);
    mostrarVector(temperaturas);
}
```

No ocurriría lo mismo si dentro de la función hacemos una nueva instancia mediante new o lo ponemos a null. Pruébalo.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

En el caso de querer pasar un elemento de un array como parámetro se debe tener en cuenta que un elemento es exactamente igual que una variable del tipo base del array, y por tanto se usará para cualquier tipo de operación y acción como el paso de parámetros.

Por ejemplo en el caso anterior podría pasar temperaturas[2] a cualquier función que admita un double como parámetro.

Fíjate también que como en este caso pasas un double, el valor del array no se verá modificado. Puedes probarlo fácilmente.

Arrays de objetos

Al igual que se crean tipos simples de arrays, también es posible tener un array de objetos. Para ellos primero hay que crear el objeto nuevo y luego colocarlos en la posición deseada del array. Veámoslo con un ejemplo:

```
import java.util.Scanner;
// En este ejemplo se hace todo en el mismo archivo, pero no
// es la práctica recomendada (clase Libro mejor en otro archivo)
class Libro {
    public String titulo;
    public String autor;
}

public class P00 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Libro libro;
        Libro[] biblioteca = new Libro[3];

        for (int i = 0; i < biblioteca.length; i++) {
            libro = new Libro();
            System.out.println("Introduce título");
            libro.titulo = sc.nextLine();
            System.out.println("Ahora el autor");
            libro.autor = sc.nextLine();

            biblioteca[i] = libro;
        }
    }
}
```

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

```

        for (int i = 0; i < biblioteca.length; i++) {
            System.out.printf("Título: %s\n", biblioteca[i].titulo);
            System.out.printf("Autor: %s\n", biblioteca[i].autor);
        }
    }
}

```

Es importante que se cree un objeto nuevo ya que si se usa uno que ya existe este cambia en todas las posiciones. Pruébalo quitando la sentencia *new Libro()* del bucle y comprueba el resultado.

El bucle for "mejorado" o "extendido" (enhanced for/foreach)

En algunos tipos de datos (tipos Iterator), como los arrays o colecciones que vemos a lo largo de este tema, se puede usar para recorrerlos un bucle for especialmente diseñado para estos tipos. Este bucle tiene la siguiente forma:

```

for (Tipo item: array){
    sentencias;
}

```

Dicho bucle **no sirve para modificar el array** pero sí para cualquier otra labor y es una forma compacta y cómoda de escribirlo. Se le suele llamar también **foreach** ya que la sentencia anterior se podría leer algo así como: "**Para cada** item del array ejecuta las sentencias".

Vemos como transformar un for clásico a un enhanced for. En el caso del ejemplo de las temperaturas, el bucle:

```

for (int i = 0; i < vector.length; i++) {
    System.out.printf("%5.1f", vector[i]);
}

```

Puede expresarse claramente de esta forma:

```

for (int i = 0; i < vector.length; i++) {
    double item = vector[i];
    System.out.printf("%5.1f", item);
}

```

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM					
	MÓDULO	Programación						CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:				
	AUTOR	Francisco Bellas Aláez (Curro)							

Resulta que el for mejorado es una forma “abreviada” de escribir la anterior expresión pues haríamos esto:

```
for (double item : vector) {
    System.out.printf("%5.1f", item);
}
```


Se leería: “Para cada item del array vector haz el printf”. Lo que hace automáticamente es crear de forma interna los índices para recorrer el array.

Lo que es importante para entender el funcionamiento de este bucle es no confundir item con un índice. Es decir, **nunca** se debería hacer dentro del bucle **vector[item]** pues no tiene sentido ya que item ya es un elemento del vector, no un índice.

Veamos en el caso de la biblioteca como mostraríamos los elementos con un bucle de estas características.

```
for (Libro lb:biblioteca) {
    System.out.printf("Título: %s\n", lb.titulo);
    System.out.printf("Autor: %s\n", lb.autor);
}
```

Dicho bucle **no sirve para modificar el array** pero sí para cualquier otra labor y es una forma compacta y cómoda de escribirlo. Veamos en el caso de la biblioteca como mostraríamos los elementos con un bucle de estas características.

	RAMA:	Informática	CICLO:	DAM					
	MÓDULO	Programación						CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:				
	AUTOR		Francisco Bellas Aláez (Curro)						

Actividad:

Ha un programa con las siguientes características (todo en el main salvo que se indique otra cosa):

a) Mediante un bucle pide 10 números enteros al usuario y almacénalos en un array.

Mediante un segundo bucle calcula la media de los números (como real).

Finalmente, mediante un tercer bucle, muestra el contenido del array y luego la media.

b) Cambia los bucles que puedas por for mejorado (deja comentado el clásico y hazlo justo debajo con el foreach).

c) Pasa la parte del cálculo de las medias a una función con parámetro el vector y que devuelva un double.

Si acabas este ejercicio puedes comenzar el ejercicio 1 del boletín.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Programación			CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)			

Arrays bidimensionales (multidimensionales)

En el apartado anterior vimos la forma de establecer mediante una única variable una lista de datos de manera que se puede acceder a cualquiera de ellos mediante un índice. En un array bidimensional el aspecto del contenedor de datos pasa a ser el de una tabla o matriz (no confundir con matrices matemáticas).

Dispone de **dos índices** lo que permite establecer **fila y columna** de acceso a dicha tabla (también se pueden ver como coordenadas).

Vamos a entenderlo con un ejemplo:

ºC	Coruña	Lugo	Ourense	Pontevedra
Lunes	15	12	14	16
Martes	16	13	12	17
Miércoles	15	12	12	16
Jueves	14	12	14	13
Viernes	17	14	15	17
Sábado	16	15	15	18
Domingo	18	16	15	19


Ejemplo Temperaturas de ciudades/semana

Para declararlos e inicializarlos tenemos las siguientes posibilidades:

```
int[][] temp = new int[7][4]; // 7 filas y 4 columnas.
```

En realidad lo anterior es un vector de 7 posiciones. Y en cada posición de ese vector contiene un vector de 4 posiciones.

Por tanto una tabla como la anterior, en Java se almacena con un aspecto similar al siguiente:

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

0	0	1	2	3
	15	12	14	16
1	0	1	2	3
	16	13	12	17
2	0	1	2	3
	15	12	12	16
3	0	1	2	3
	14	12	14	13
4	0	1	2	3
	17	14	15	17
5	0	1	2	3
	16	15	15	18
6	0	1	2	3
	18	16	15	19

A continuación se presentan otras formas de inicializar arrays bidimensionales:

```
int matriz[][] = new int[2][3]; // Corchetes en el nombre
```

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

```
// Array predefinido al declarar.
int[][] tabla = {
    {15,12,14},
    {12, 1, 3},
    {14, 0,16}
};

// Array predefinido en un punto distinto al a declaración.
char[][] letras;
letras = new char[][]{
    {'a','b','c'},
    {'d','e','f'}
};
```

Los casos anteriores son matrices con todas las filas del mismo tamaño. También pueden tener tamaño distinto aunque se use menos. Para ellos la inicialización sería así:

```
int[][] b= {
    {-5, 7, 7, 9},
    {1, 2},
    {4, 3, 0}
};

int[][] a = new int[3][]; // 3 filas
a[0] = new int[2]; // Fila 0 tiene 2 columnas
a[1] = new int[7]; // Fila 1 tiene 7 columnas
a[2] = new int[5]; // Fila 2 tiene 6 columnas
```

Para recorrer estas tablas es necesario utilizar un doble bucle anidado (habitualmente se usa for o for mejorado) teniendo como límites el número de filas y el de columnas. Para conseguir estos límites se puede usar *length* de la siguiente manera:

```
System.out.println("Número de filas: "+temp.length);
System.out.println("Número de columnas: "+temp[0].length);
```

El número de columnas lo obtenemos del tamaño (*length*) de la fila 0, pero realmente se podría obtener de cualquiera de las filas (que es como se hace en los bucles siguientes mediante *i*).

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Teniendo esto en cuenta para rellenar las celdas de una tabla con datos aleatorios sería de la siguiente forma:

```
for (int i = 0; i < temp.length; i++) {
    for (int j = 0; j < temp[i].length; j++) {
        temp[i][j] = (int) (Math.random() * 40 - 10);
    }
}
```

Y para leer los datos y (en este caso) mostrarlos por pantalla en forma de tabla podríamos usar también un doble bucle for normal:

```
for (int i = 0; i < temp.length; i++) {
    for (int j = 0; j < temp[i].length; j++) {
        System.out.printf("%3d", temp[i][j]);
    }
    System.out.println();
}
```

o un doble bucle for mejorado:

```
for (int[] fila : temp) {
    for (int valor : fila) {
        System.out.printf("%3d", valor);
    }
    System.out.println();
}
```

Existe la posibilidad de realizar arrays de más dimensiones, simplemente consiste en añadir más "corchetes" en la definición. Por ejemplo:

```
int [][][] array3D;
```

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Las clases Arrays y Array.

Esta clase contiene métodos estáticos que permiten una serie de operaciones y manipulaciones de arrays de forma sencilla. En este apartado simplemente citaré dicha clase para que **no se use como constructor** de un array por error.

```
Array[][] temp = new Array(); // N0000000000000000!!!!!!!!!!!!!!!
```

Realmente tiene una serie de métodos estáticos que pueden ser útiles en un momento dado como métodos de ordenación o para copiar una sección del array.

Para más información accede a la página de documentación:

<https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html>

<https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html>

donde se ven todas las sobrecargas existentes para cada una de las funciones.

Actividad:

Realiza una función a la cual se le pasan dos parámetros enteros y cree un array bidimensional de números reales (aleatorios entre 20 y 30) del tamaño indicado por los parámetros (filas y columnas) y lo devuelva.

Realiza también otra función a la que se le pasa un array bidimensional de doubles y la muestra por pantalla indicando el número de fila y el de columna como cabeceras. Los datos muestra los con 2 decimales. Trata de hacerla con for mejorado.

En el main crea dos arrays mediante la primera función (de 4x3 y de 3x4). Luego muéstralos mediante la segunda función.

Si acabas este ejercicio puedes hacer el ejercicio 2 del boletín.

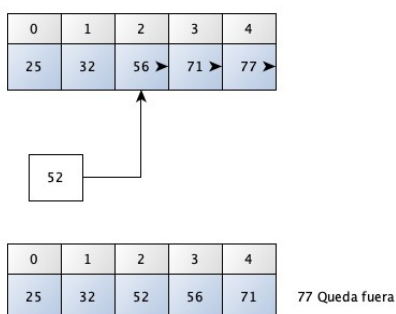
<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Introducción a las colecciones (ArrayList)

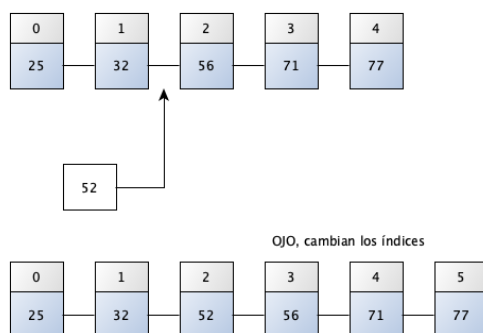
En las colecciones, a diferencia de los arrays, disponemos de una lista de elementos pero cuyo tamaño puede evolucionar a lo largo de la ejecución de un programa de una forma más sencilla. Es posible insertar y eliminar elementos en distintas posiciones.

Se puede ver un array como una estantería con divisiones de forma que entre cada dos divisiones cabe un dato. Esa estantería tiene un tamaño fijo, es decir, una cantidad fija de divisiones que no puede cambiar.

No puedo añadir sin eliminar algún dato ya existente. Si quiero insertar en medio tengo que desplazar elementos pero siempre teniendo en cuenta que al final se ha de eliminar el dato del final del vector (o estantería).



Sin embargo una colección puede verse como una tren con vagones. Si quiero insertar un nuevo dato, simplemente desengancho los vagones entre los cuales va el nuevo dato y meto ahí en medio el nuevo vagón. El caso de eliminación es similar.



COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Por tanto un array en una estructura de tamaño predeterminado y simplemente se meten y quitan datos en distintas posiciones, mientras que una colección es una estructura dinámica en la cual se crean o se destruyen contenedores para los datos.

La desventaja de las colecciones frente a los arrays es que son menos eficientes.

Existen diversos tipos de colecciones en Java. En este apartado introduciremos los de la clase **ArrayList** que son, quizá, los más sencillos y genéricos. Sin embargo al final del tema existe un apéndice con una breve descripción de otros tipos por si alguien desea profundizar.

Para declarar e instanciar un objeto ArrayList se realiza de la siguiente forma:

```
ArrayList<T> nombreColección = new ArrayList<T>();
```

Siendo **T** la clase base de los objetos que vamos a guardar en el **ArrayList**. Por ejemplo:

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

Es decir, de forma similar a crear cualquier otro objeto. La clase base *T* puede ser una clase creada por nosotros.


Se requiere realizar el **import java.util.ArrayList**.

Y es necesario indicar que **no se pueden hacer colecciones de tipo primitivos**: int, boolean, char, double, etc... ya que solo se admiten colecciones de objetos. Si fueran necesario habría que usar una de las **clases wrapper** comentadas en el primer tema que sirven de "clase" para los tipos primitivos. Entonces si se desea realizar una colección de enteros no se puede hacer esto:

```
ArrayList<int> numeros=new ArrayList<int>(); // N00000000
```

si no que habría que hacer:

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


	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Métodos de colecciones

Una vez que tenemos la colección creada (supón que tiene de nombre **col**) tenemos una serie de métodos que nos permiten manejarla. Veamos un resumen de algunos y luego los pondremos en práctica:

col.add(elemento):

Añade un nuevo elemento a la colección (al final).

col.add(posición, elemento):

Inserta un nuevo elemento en la posición indicada.

col.size():

Devuelve el nº de elementos de la colección.

col.remove(posición):

Elimina el elemento situado en la posición indicada en el parámetro. Devuelve dicho objeto.

col.get(posición):

Devuelve el valor del dato de la posición indicada.

col.set(posición, elemento):

Cambia el dato que hay en posición con el elemento del parámetro.

col.contains(elemento):

Devuelve true si el elemento está en la colección.

col.indexOf(elemento):

Devuelve el índice del objeto indicado. Devuelve -1 si no encuentra el elemento buscado.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

En concreto del **contains** y del **indexOf** hablaremos en un futuro tema. Por ahora lo puedes usar solo con clases ya definidas (String, Integer, etc) pero te va a dar problemas al usarlo con clases definidas por ti como Perro o Libro. Lo solucionaremos en el siguiente tema.

Ejemplo 1: Colección de enteros

Veamos a continuación con ejemplos como se usarían la colección y sus métodos.

```
// Instaciamos colección vacía
ArrayList<Integer> numeros = new ArrayList<Integer>();

// Añadimos elementos
numeros.add(3);
numeros.add(5);
numeros.add(9);

// Insertamos en posición 1
numeros.add(1, 10);

// Mostramos con for clásico
for (int i = 0; i < numeros.size(); i++) {
    System.out.printf("%5d", numeros.get(i));
}
System.out.println();

// Modificamos un elemento
numeros.set(0, 22);

// Eliminamos el elemento de la posición 2 (Es el tercero)
numeros.remove(2);

// Mostramos con foreach
for (int numero : numeros) {
    System.out.printf("%5d", numero);
}
System.out.println();
```

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM					
	MÓDULO	Programación						CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:				
	AUTOR	Francisco Bellas Aláez (Curro)							

Ejemplo 2: Colección de objetos definidos por el programador

A continuación una colección más compleja a partir de una clase creada por nosotros. Si se ha entendido lo anterior se puede seguir esto sin problema leyendo los comentarios de cada conjunto de instrucciones.

Lo primero que haremos es crear una clase sencilla con simplemente dos propiedades y un constructor que las inicialice:

```
public class Libro {
    public String titulo;
    public String autor;

    public Libro(String titulo, String autor) {
        this.titulo = titulo;
        this.autor = autor;
    }
}
```

a continuación en la clase principal que ejecutaremos introducimos la siguiente función de visualización de los elementos de la colección para facilitarnos luego el programa principal. Se usa un foreach para recorrerla.

```
public static void mostrarBiblioteca(ArrayList<Libro> biblioteca){
    System.out.println("Libros en la biblioteca");
    System.out.println("=====");
    for (Libro libro : biblioteca) {
        System.out.printf("Titulo:%s\nAutor:%s\n\n", libro.titulo,
                           libro.autor);
    }
}
```

En el main escribimos las siguientes líneas, las ejecutamos y las analizamos:

```
ArrayList<Libro> biblioteca = new ArrayList<Libro>();
```

```
//Declaramos una variable tipo Libro que añadiremos a la colección
Libro lb = new Libro("El juego de Ender", "Orson Scott Card");
biblioteca.add(lb);
```

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

```
// Añadimos tres libros más. Ya no es necesario declarar, sólo instanciar.
    lb = new Libro("Proyecto Hail Mary", "Andy Weir");
    biblioteca.add(lb);

    lb = new Libro("Por no mencionar al perro", "Connie Willis");
    biblioteca.add(lb);

    // Se permite pasar el objeto como parámetro si no nos hace falta
    // la referencia para otras tareas.
    biblioteca.add(new Libro("Dune", "Frank Herbert"));

// Añadimos un nuevo libro ahora en la posición 2
    lb = new Libro("El éxodo de los gnomos", "Terry Prachett");
    biblioteca.add(2, lb);

// Mostramos el tamaño de la colección
    System.out.println("El tamaño de la colección es: " + biblioteca.size());

// Visualizamos el contenido de la biblioteca
    mostrarBiblioteca(biblioteca);
```

En esta primera parte se puede ver como crear la colección y como insertar elementos en la misma y obtener su tamaño. Añadimos estas líneas:

```
//Eliminamos el elemento en la posición 2
    Libro eliminado = biblioteca.remove(2);
    mostrarBiblioteca(biblioteca);
    System.out.println("Se ha eliminado el libro: " + eliminado.titulo);

//El tamaño de la colección ahora
    System.out.println("El tamaño de la colección es: " + biblioteca.size());

//Obtenemos el elemento 0 y lo mostramos
    lb = biblioteca.get(0);
    System.out.printf("Titulo:%s\nAutor:%s\n\n", lb.titulo, lb.autor);
```

Vemos en las líneas previas como eliminar y como obtener un elemento sin eliminarlo.

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Cadenas

Una cadena como ya sabemos es una secuencia de caracteres tratadas como una unidad. Internamente se gestiona como si fuera un **vector de caracteres**. De hecho en Java se puede leer cada carácter por separado con el método **charAt**.

En cualquier lenguaje de programación suele haber algún tipo de sistema de proceso de cadenas por ser una variable muy cercana al ser humano. En el caso de Java está principalmente la clase **String** que será en la que nos centraremos, pero existen otras clases como **StringBuilder** y **Character**.

La clase String

Esta clase permite representar cadenas de caracteres **inmutables**, es decir, que no pueden cambiar internamente porque no hay métodos en la clase que lo hagan. Si se quiere cambiar el contenido de una variable tipo String hay que cambiarla por por otro String totalmente nuevo.

En caso de querer trabajar con cadenas que cambien, el tipo mutable es **StringBuilder**.

En el caso de String es notable que nunca se hace new (aunque se puede). Puede parecer que es una excepción pero no es así. Siempre la inicializamos asignando un objeto literal predefinido, es decir, un literal cadena. Por lo cual ya tenemos una referencia apuntando al objeto

A continuación veremos algunos métodos útiles de la clase String. Te en cuenta que al ser una tipo inmutable, **estos métodos no cambian el String original** si no que ese queda tal cual y el método, si fuera necesario, devuelve una versión modificada del String. Suponiendo que la variable **cadena** tiene un String determinado, los métodos son:

cadena.length():

Devuelve el tamaño de la cadena

cadena.charAt(índice):

Devuelve el carácter en la posición indicada.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

cadena.toLowerCase():

Devuelve la cadena toda en minúsculas.

cadena.toUpperCase():

Devuelve la cadena toda en mayúsculas.

cadena.equals(cadena2):

Devuelve true si cadena y cadena2 son exactamente iguales. Si no fuera así devuelve false.

cadena.equalsIgnoreCase(cadena2):

Compara si cadena y cadena2 son iguales independientemente que estén en mayúsculas o minúsculas. Devuelve el booleano correspondiente.

cadena.startsWith(cadena2):

Devuelve true si cadena comienza por cadena2.

cadena.endsWith(cadena2):

Devuelve true si cadena termina por cadena2.

cadena.indexOf(subcadenaOCaracter):

Devuelve la posición de la primera aparición de la subcadena o del carácter (sobrecarga). También tiene una sobrecarga que permite introducir a partir de qué carácter queremos buscar.

cadena.lastIndexOf(subcadenaOCaracter):

Como la anterior pero devuelve la aparición de la última aparición.

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

cadena.substring(inicio, fin):

Devuelve un fragmento de subcadena que empieza en inicio y va hasta fin (**esta posición no incluida**). Existe una sobrecarga con un único parámetro que es el inicio y va hasta el final de la cadena.

Concatenación:

Para unir cadenas podemos usar el operador +, += o el método concat. Por ejemplo:

```
cad1 += cad2; //Coloca cad2 y a continuación cad1 y lo guarda en cad1.
cad1 = cad1 + cad2; //Lo mismo que en el caso anterior.
cad1.concat(cad2); //Lo mismo que en el caso anterior.
cad1 = cad2 + cad1; //Coloca cad2 y a continuación cad1 y lo guarda en cad1.
```

Es interesante ver la diferencia entre el primer y el último caso. En el primero concatena a continuación, en el último concatena al principio.

cadena.trim():

Devuelve la cadena eliminando los espacios extremos. Solo el Unicode 0x20.

cadena.strip():

Como trim() pero funciona con todos los códigos de espacio Unicode.

cadena.split(separador):

Devuelve un array de strings que contiene en cada posición una parte de la cadena según el separador. Por ejemplo si se le pasa como separador un espacio, coge las palabras de una frase y cada una la mete en una posición del array de Strings.

Ojo porque si quieres separar por alguno de estos caracteres (metacaracteres):

`\ , ^ , $, . , | , ? , * , + , (,) , { , } , [`

Debes anteponerle `\\` ya que tienen cierta utilidad ya que se permiten las llamadas expresiones regulares. Estas pueden usarse para hacer splits más complejos.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Prueba estos dos ejemplos a ver si eres capaz de deducir cómo funcionan:

```
String cad = "hola?adios.Que tal@estás,tú";
for (String c : cad.split("[, ?.@]")) {
    System.out.println(c);
}
System.out.println();

String animales = ";01Gallina;02Perro;030rnitorrinco;14Cebra;300kapi";
String[] v = animales.split(";[0-9][0-9]");
for (String item : v){
    System.out.println(item);
}
```

Puedes leer más en:

<http://puntocomnoesunlenguaje.blogspot.com/2013/07/ejemplos-expresiones-regulares-java-split.html>

En este artículo se usa para búsquedas pero las variables tipo Pattern puedes usarlas también en el split.

String.format(cadena de formateo, variables o expresiones):

Funciona exactamente **igual que el printf** pero con la salvedad que en vez de mostrar la cadena formateada por pantalla la **devuelve** para poder trabajar con ella sin imprimirla en pantalla.

Por tanto lo visto para dicha función printf, sirve también aquí. Aprovechando que estamos trabajando con Strings conviene añadir un formateador muy interesante y es el uso del punto en % (como hacíamos con los reales), que sirve para mostrar un fragmento.

Por ejemplo **%.6s** corta a 6 caracteres si el string es mayor. Con una combinación como **%6.6s** creas un ancho fijo de 6 caracteres con justificación a la derecha.

Puedes profundizar más en el formateo en este enlace, verás que es realmente versátil:

<https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html>

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Otras clases para texto


StringBuilder: Permite manejo de cadenas mutables, es decir, que pueden cambiar. Salvo que se hagan muchas concatenaciones no suele ser demasiado útil esta clase al ser menos eficiente. Más información en:

<https://docs.oracle.com/javase/tutorial/java/data/buffers.html>

Character: Con un montón de métodos para el manejo de caracteres. Más información en:

<http://docs.oracle.com/javase/7/docs/api/java/lang/Character.html>

https://www.tutorialspoint.com/java/java_characters.htm

	RAMA:	Informática	CICLO:	DAM					
	MÓDULO	Programación						CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:				
	AUTOR		Francisco Bellas Aláez (Curro)						

Ejercicio (Para validación):

Declara e inicializa una variable de tipo String con la frase: "Jar-Jar is the Big Boss"

Utilizando las funciones citadas anteriormente y viendo como se usan mediante la documentación:

<https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>

En el main realiza las siguientes tareas (deben funcionar aunque se cambie la frase):

- Muestra la longitud de la cadena
- Muestra el primer carácter, el último y comprueba lo que pasa al acceder al carácter de la posición 100.
- Crea una segunda cadena a partir de la primera pasándola a mayúsculas y observa el resultado de equals y equalsIgnoreCase.
- Muestra la cadena en minúsculas.
- Comprueba si la cadena acaba por "Boss" y por "Jar".
- Muestra la posición de la primera y la última vez que aparece la palabra "Jar".
- Crea otra variable String a partir del fragmento de la cadena anterior que empieza en la posición 7 y acaba en la 14 (ambos incluidos).
- Quita los espacios de los extremos de la anterior cadena creada y muéstrala.
- Crea un array de Strings con las palabras de la primera cadena (los separadores son el espacio y el guion).
- Muestra cada palabra del array anterior en una línea ocupando 3 caracteres (cortala si es mayor). Esto no lo hagas con substring si no jugando con el formato de printf.

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Apéndice I: Otras colecciones

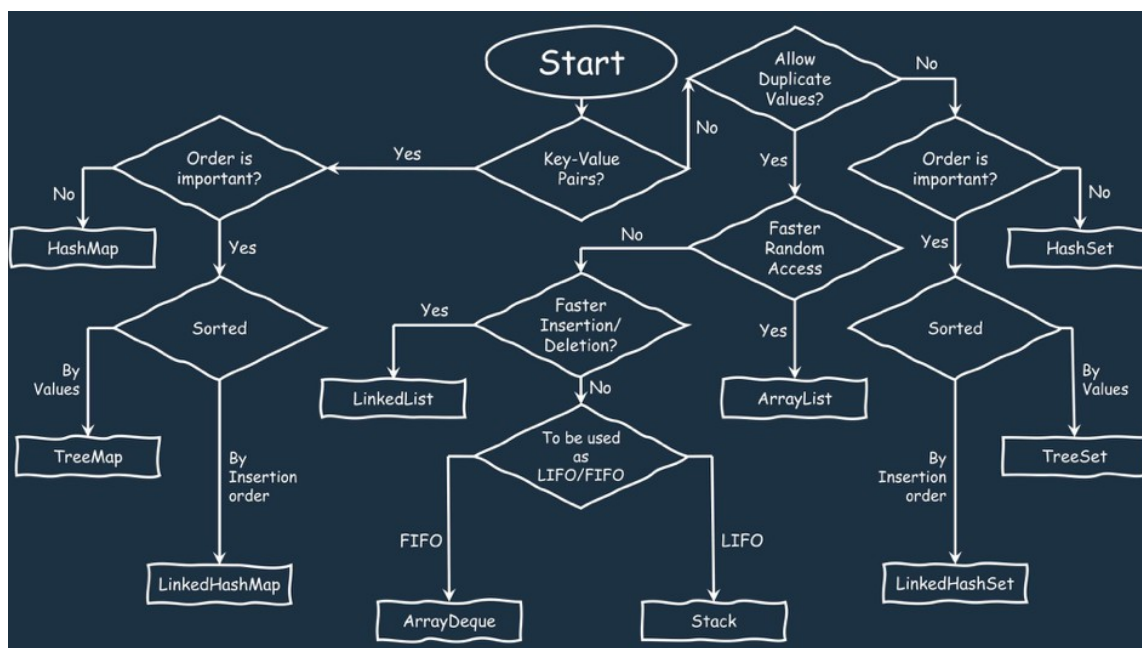
En este tema hemos visto las colecciones genéricas tipo ArrayList, sin embargo Java dispone de otros tipos de colecciones que son muy prácticas dependiendo de la funcionalidad que se busque. No vamos a profundizar en ellas salvo que en los próximos temas nos haga falta alguna, pero siempre es bueno conocerlas para buscar luego más información en alguna referencia técnica. Algunas de estas colecciones son:

Set y derivados: El concepto es similar al de conjunto matemático. Es decir, es una colección que no tiene duplicados.

Maps o Diccionarios: diccionario o tabla hash de forma que cada posición se representa con una clave y contiene un valor determinado.

FIFO/LIFO: Es una estructura de datos tipo First In First Out/Last in First Out también denominados colas o pilas respectivamente.

Si quieres más información sobre estas colecciones mira este esquema y busca información sobre cada una de las colecciones que te pudiera interesar:



<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR	Francisco Bellas Aláez (Curro)						

Apendice II: parámetros opcionales

Es posible crear en Java una función a la cual se le pasa un número indeterminado de parámetros. Para ello en la función se usa la sintaxis

`TipoDato ... params`

En este caso, `params` se puede ver dentro de la función como un array unidimensional del tipo *TipoDato* y desde fuera de la función como que puedo pasar varios parámetros del tipo *TipoDato*. Veámoslo con un ejemplo:

```
public class Opcionales {
    public static int suma(int ... numeros) {
        int resultado = 0;
        for (int i = 0; i < numeros.length; i++)
            // Puede usarse un for mejorado
            resultado += numeros[i];
        return resultado;
    }

    public static void main(String[] args) {
        int x = 5;
        int total = suma(2, -45, x, 5 * x);
        System.out.println(total);
        System.out.println(suma(12, -2, 54, 1, 81));
        System.out.println(suma(3, 6, 21));
    }
}
```

La condición principal es que la definición de los parámetros opcionales se haga siempre al final de la definición de parámetros.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Apéndice III: Parámetros desde la línea de comandos

Hasta ahora hemos escrito la función *main* con un parámetro que no hemos utilizado ni sabemos su utilidad. Dicho parámetro es un array unidimensional de cadenas (String[]) y su utilidad es que se pasen parámetros desde la línea de comandos al programa.

Viéndolo desde la perspectiva del entorno gráfico es el equivalente a cuando realizamos doble clic sobre un documento de texto y este se abre con el bloc de notas. Realmente lo que está ocurriendo es que al realizar doble clic sobre un archivo de texto es como si escribiéramos en la línea de comandos en el caso de Ubuntu:

```
$ gedit nombreadchivo.txt
```

o en el caso de Windows:

```
C:\> notepad nombreadchivo.txt
```

Es decir que ejecutamos el programa y le pasamos un parámetro. Si lo hiciéramos dentro de nuestra aplicación es como si escribiéramos

```
main({"nombreadchivo.txt"});
```

En el caso de pasar varios parámetros en línea de comandos, estos se separan por espacios. Por ejemplo si en línea de comandos escribimos lo siguiente:

```
(Linux) $ cp archivo directorio
(Windows) C:\>copy archivo directorio
```

sería como poner

```
main({"archivo","directorio"});
```

Es decir, los espacios en la línea de comando son como las comas en la definición de un array: separadores de elementos. Y a la función *main* se le pasa un array que contiene los elementos definidos en la línea de comandos a continuación del nombre del programa.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Con lo explicado se debería entender el funcionamiento del siguiente programa ejemplo:

```
public static void main(String[] args) {
    int i;

    if (args.length == 0) {
        System.out.println("No ha pasado ningún argumento. Se debe usar:");
        System.out.println("\t-M: Paso a mayusculas\n\t-m: Paso a minusculas");
        System.out.println("y a continuacion la cadena a formatear");
    } else
        if (args[0].equals("-m")) {
            for (i = 1; i < args.length; i++)
                System.out.print(args[i].toLowerCase() + ' ');
            System.out.println();
        } else
            if (args[0].equals("-M")) {
                for (i = 1; i < args.length; i++)
                    System.out.print(args[i].toUpperCase() + ' ');
                System.out.println();
            } else {
                System.out.println("Argumento no valido. Se debe usar:");
                System.out.println("\t-M: Paso a mayusculas");
                System.out.println("\t-m: Paso a minusculas");
            }
}
```

Para probarlo, en la línea de comando:

```
$ java Argumentos -M Java, Java everywhere...
```

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Apéndice IV: Escribir emojis en Java.

Como ya se ha planteado, para escribir caracteres unicode se puede usar el carácter de escape \uXXXX. Pero el carácter mayor que se puede escribir de esa manera es

\uFFFF.

Los emojis (y otros caracteres por encima del code point unicode U+FFFF) se pueden escribir o usando varios \u para expresarlo en UTF16 como en este ejemplo (Tiene que soportarlo la consola o interfaz gráfica):

```
// El code point U+1F600 es en UTF-16 0xD83D 0xDE00
System.out.println("\uD83D\uDE00");
```

O se puede usar un constructor de String donde se le puede dar un array de enteros que cada uno es un code point Unicode que representa un emoji (u otro carácter). Aquí un ejemplo que muestra los 50 primeros emojis:

```
// Se crea un array con los unicode de los emojis deseados.
// En este caso simplemente los 50 primeros.
int[] emojis = new int[50];
for (int i=0; i<emojis.length; i++) {
    emojis[i]=0x1F600+i; // En unicode la lista de emojis empieza en U+1F600
}
//Constructor de String que convierte de enteros a utf-16 (necesario para
emojis)
String texto = new String(emojis, 0, emojis.length);
System.out.println(texto);
```

Para que esto funcione debe soportarlo el interfaz de usuario (consola utilizada o la librería gráfica correspondiente).

Hay conversores de unicode a utf en internet como <https://r12a.github.io/app-conversion/>