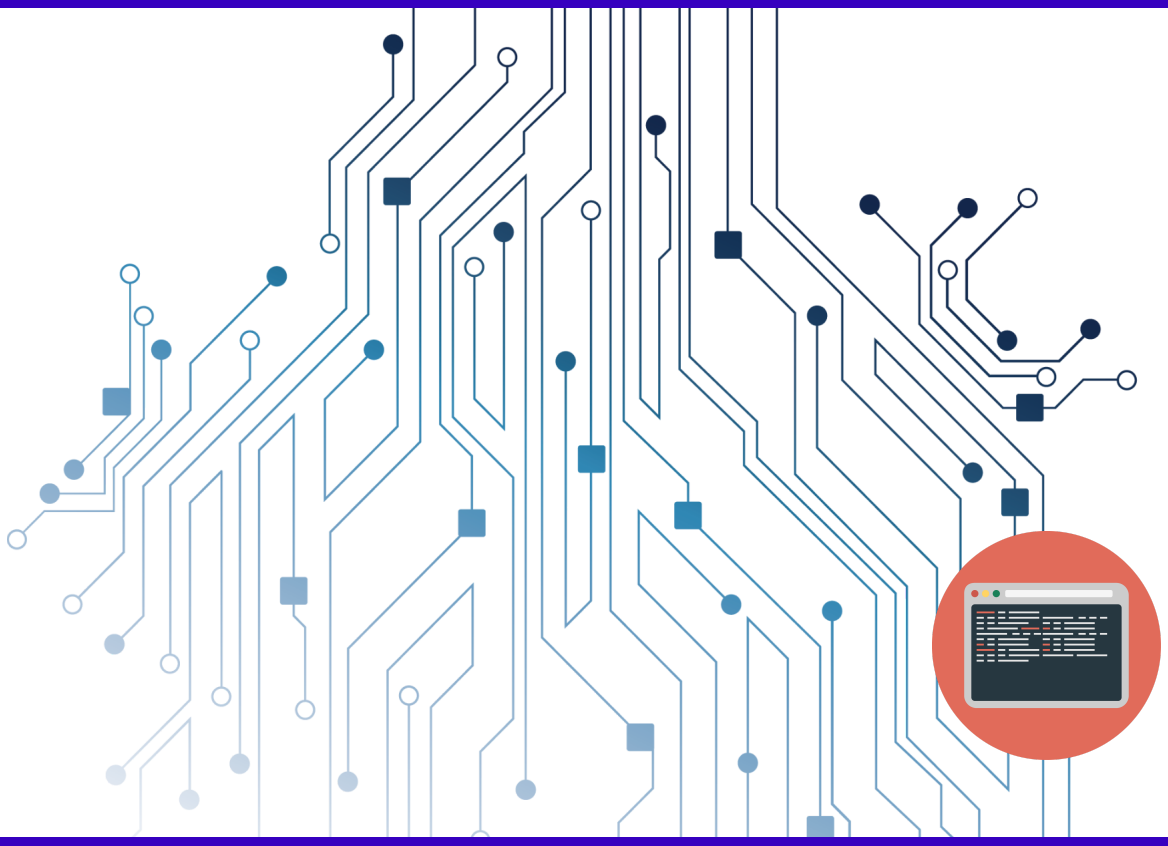
CFGS APLICACIONES MULTIPLATAFORMA - PROGRAMACIÓN

UNIDAD 5

COLECCIONES



DIEGO VALERO ARÉVALO
BASADO EN APUNTES Y EJERCICIOS PROPORCIONADOS POR
Mª CARMEN DÍAZ GONZÁLEZ - IES VIRGEN DE LA PALOMA

U5 - **COLECCIONES**
ÍNDICE

5.1 - COLECCIONES

Las **colecciones** son **objetos que almacenan referencias a otros objetos**, dicho de otra manera, podemos considerarlos **"arrays de objetos"**.

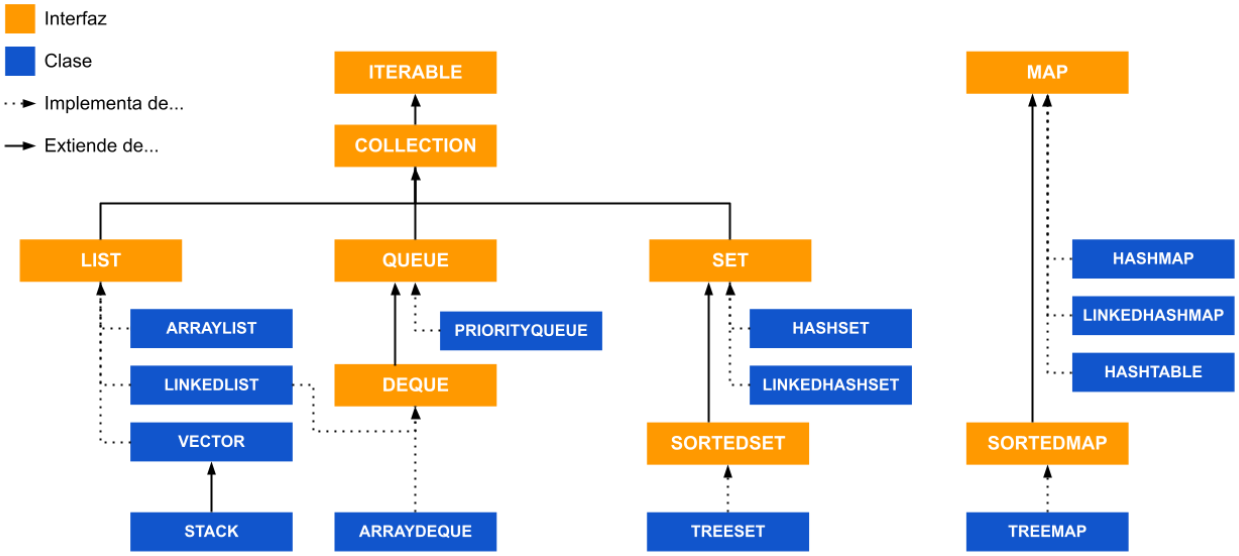
A diferencia de los arrays que conocemos, las colecciones son **dinámicas**, que quiere decir que **no trabajan con un tamaño fijo** y permiten añadir y eliminar objetos durante la ejecución del programa.

Java incluye en el paquete `java.util` un amplio conjunto de clases para la creación y tratamiento de colecciones. Todas ellas proporcionan una serie de métodos para realizar las operaciones básicas sobre una colección, como son añadir objetos, eliminarlos, obtenerlos, localizarlos, iterar a través de la colección,...

En esta unidad veremos las siguientes interfaces y sus derivaciones:

List	Set	Map

De las cuáles puedes ver su relación en el siguiente diagrama:



Como ves, aunque `MAP` sea considerada una colección de objetos, **no extiende** de la interfaz `COLLECTION`, pero se suele añadir a este repertorio por su funcionalidad.

Todas las clases que ves en el diagrama cumplen el mismo propósito, pero cada una varía y tiene sus características. Vamos a ir viendo poco a poco muchas de ellas.

5.2 - ARRAYLIST

Los `arraylist` son **colecciones dinámicas** que implementan **listas de tamaño variable**, cuyo tamaño por defecto es `0`. Es similar a un array con las ventajas de que **su tamaño crece dinámicamente**

conforme se añaden elementos (no es necesario fijar su tamaño al crearlo) y permite almacenar datos y objetos de cualquier tipo.

Aunque los `arraylist` permiten combinar tipos de datos diferentes, **no es recomendable** para una buena organización y manipulación, ya que esta colección trata a sus elementos como **objetos**, ignorando si son tipos primitivos o no. Para ello podemos especificar el tipo de objetos del `arraylist` con el modificador `<tipo>` que puedes ver en el siguiente apartado: **5.2.1 - *Declarar un Arraylist***.

>> 5.2.1- DECLARAR UN ARRAYLIST

Antes de poder declarar cualquier `arraylist`, debemos **importar la clase**. Después podremos declararla como hacemos con el resto:

```
import java.util.Arraylist;

ArrayList<tipo> nombreAL = new ArrayList<tipo>();
```

`ArrayList`	Nuestro objeto será de tipo `ArrayList`.
`<tipo>`	Aunque no es obligatorio, es recomendable **especificar el tipo de objetos** que contendrá el `arraylist`. Podemos **no ponerlo**, aceptar **cualquier tipo con `<E>`** (**tipo genérico**) o **especificar directamente el tipo**.
`nombreAL`	Asignamos un **nombre** al mismo.
`new`	El operador `new` asigna durante la ejecución del programa un **espacio de memoria al objeto**, y almacena esta **referencia** en una variable.
`ArrayList<tipo>();`	La clase en la que se va a basar el objeto, que será `ArrayList`. Si especificamos el tipo de objetos en la declaración, es recomendable colocarlo aquí también.

>> 5.2.2- AÑADIR DATOS A UN ARRAYLIST

`nombreAL.add(dato);`	Simplemente el método `.add` y lo que queramos añadir entre los paréntesis. Acepta cualquier tipo de primitivas e incluso podemos declarar objetos directamente.

EJEMPLO

.java

```
1  listado.add(-25);
2  listado.add(3.4);
3  listado.add('a');
4  listado.add("hola");
5  listado.add(new Persona("28751533Q", "María", "González Collado", 37));
6
7  //E incluso declarar objetos y añadirlos con su variable:
8  Persona p = new Persona("16834954R", "Raquel", "Dobón Pérez", 62));
9  listado.add(p);
```

>> 5.2.3- MÉTODOS DE ACCESO Y MANIPULACIÓN DE UN ARRAYLIST

La clase **`ArrayList`** ofrece muchos métodos diferentes para poder hacer operaciones con sus elementos. Vamos a ver los más utilizados:

`int .size();`	Devuelve el número total de elementos del **`arraylist`**.
`E .get(index);`	Devuelve la referencia al elemento en la posición **`index`**.
`void .clear();`	Vacía el **`arraylist`** y establece el tamaño a `0`.
`boolean .isEmpty();`	Devuelve true si el **`arraylist`** está vacío.
`boolean .add(dato);`	Como ya hemos visto, inserta un dato en la lista, siempre en la **última posición**.
`void .add(index, dato);`	Inserta el **`dato`** en la posición **`index`** de la lista. D**esplaza una posición** todos los demás elementos de la lista (**no sustituye ni borra otros elementos**).
`void .set(index, dato);`	Sustituye el elemento en la posición **`index`** por el nuevo **`dato`**.
`boolean .contains(dato);`	Busca el **`dato`** en la lista y devuelve **true** si existe. Este método utiliza **`.equals()`** para comparar los objetos.
`int .indexOf(dato);`	Busca el **`dato`** en la lista, empezando por el principio, y devuelve el **índice** dónde se encuentre. Devuelve **`-1`** si **no existe**. Este método utiliza **`.equals()`** para comparar los objetos.
`int .lastIndexOf(dato);`	Igual que el anterior pero **devuelve el índice de la última coincidencia**, ignorando las anteriores.
`E .remove(index);`	**Elimina** el elemento en la posición **`index`** y lo devuelve.
`boolean .remove(dato);`	**Elimina la primera ocurrencia** de **`dato`** en la lista. Devuelve **`true`** si lo ha encontrado y eliminado, y si no, **`false`**. Este método utiliza **`.equals()`** para comparar los objetos.
`void .remove(index);`	**Elimina** el objeto de la lista que se encuentra en la posición **`index`**. Es más rápido que el método anterior ya que no necesita recorrer toda la lista.

Puedes encontrar muchas más información consultando el **API** de Java de la clase **`ArrayList`**.

>> 5.2.4- RECORRER UN ARRAYLIST

Hay varias maneras diferentes en las que se puede **iterar** (recorrer) una colección del tipo ArrayList. La más fácil es como recorrer un array normal, con un bucle for, pero hay otras dos maneras útiles: a través de un bucle **`for-each`** o un **`iterador`**.

· 5.2.4.1- BUCLE FOR-EACH

Este tipo de bucle es una manera resumida de **`for`** que no usa contadores ni límites, simplemente se limita a acceder a cada uno de los elementos existentes en una colección.

```
for(tipo v : colección){
   instrucciones...
}
```

`tipo`	Debemos especificar de qué **tipo** son **los elementos** de la colección que vamos a recorrer.
`v`	Damos un **nombre** para poder **acceder a las posiciones** (se suele usar `v`). Esto actúa de medio para **devolver el valor** de cada elemento.
`colección`	Establecemos la **colección** que queremos recorrer.

EJEMPLO

.java

```java
ArrayList<String> miLista = new ArrayList<String>();

miLista.add("hola");
miLista.add("casa");
miLista.add("árbol");

for(String v : miLista){
    System.out.println(v+", ");
}
```

Console

```
hola, casa, árbol,
```

· 5.2.4.1- ITERADOR

Usando un objeto `Iterator` que permite **recorrer listas como si fuese un índice**. **Se necesita importar la clase antes**.

```java
import java.util.Iterator;

Iterator nombreIterator = colección.iterator();
```

`Iterator nombreIterator`	Creamos el objeto de tipo `Iterator` y le damos un nombre.
`colección.iterator();`	Accedemos a la colección que queremos iterar y le añadimos el método `.iterator()`.

Tiene dos métodos principales:

`.hasNext()`	**Verifica** si hay más elementos después del actual.
`.next()`	**Devuelve** el objeto actual y **avanza al siguiente**.

EJEMPLO

.java

```java
import java.util.Iterator;

ArrayList<String> miLista = new ArrayList<String>();
Iterator iterador = miLista.iterator();

miLista.add("hola");
miLista.add("casa");
miLista.add("árbol");

while(iterador.hasNext()){
    System.out.println(iterador.next()+", ");
}
```

Console

```
hola, casa, árbol,
```

Para recorrer el `arraylist` con un `iterator`, comprobamos si hay más elementos después del actual con `.hasNext()`, y si es así, imprime el actual y mueve el iterador al siguiente de la lista con `.next()`.

5.3 - HASHMAP

Un **`HashMap`** es una **colección dinámica** que no tiene posiciones, sino que guarda los elementos en función de una **clave** y un **valor** que le corresponde. El acceso a estos es a través de la clave, que **no pueden repetirse ni ser nulas**, al contrario que los valores.

> Los **`HashMap`** no guardan el orden ni garantizan el orden de inserción cuando los recorremos.

>> 5.3.1- DECLARAR UN HASHMAP

Antes de poder declarar cualquier **`HashMap`**, debemos **importar la clase**. Después podremos declararla como hacemos con el resto:

```
import java.util.HashMap;

HashMap<tipoClave, tipoValor> nombreHM = new HashMap<tipoClave, tipoValor>();
```

`HashMap`	Nuestro objeto será de tipo **`HashMap`**.
`<tipoClave, tipoValor>`	Debemos **especificar el tipo de las claves y de los valores** que contendrá el **`HashMap`**. para que el programa sepa cómo trabajar con ellos. > Los **`HashMap`** **no aceptan tipos primitivos**, por lo que debemos usar **wrappers** (clases que representan primitivos, como **`Integer`**).
`nombreHM`	Asignamos un **nombre** al mismo.
`new`	El operador **`new`** asigna durante la ejecución del programa un **espacio de memoria al objeto**, y almacena esta **referencia** en una variable.
`HashMap<tipoClave, tipoValor>();`	La clase en la que se va a basar el objeto, que será **`HashMap`**. Se añaden también los tipos de las claves y los valores.

>> 5.3.2- AÑADIR DATOS A UN HASHMAP

`nombreHM.put(clave, valor);`	Simplemente con el método **`.put`** y añadimos **la clave y su valor** entre los paréntesis.

EJEMPLO

`.java`

```
1  import java.util.HashMap;
2
3  HashMap<Integer, String> miHashMap = new HashMap<Integer, String>();
```

```
4
5  miHashMap.put(1, "Primer valor");
```

>> 5.3.3- MÉTODOS DE ACCESO Y MANIPULACIÓN DE UN HASHMAP

La clase **`HashMap`** ofrece muchos métodos diferentes para poder hacer operaciones con sus elementos. Vamos a ver los más utilizados:

`.get(clave)`	Este método nos devuelve el **valor de la clave** que le pasemos. Si no existe la clave, devuelve **`null`**.
`.put(clave, valor)`	Como ya hemos visto, con esto **añadimos una clave y su valor** al **`HashMap`**. Si ya existe la clave, **sustituye el valor**.
`.remove(clave)`	**Elimina la clave y su valor** del **`HashMap`**. Si no existe la clave, devuelve **`null`**.
`.keySet()`	Devuelve un conjunto **`set`** con **todas las claves**.
`.values()`	Devuelve una **colección** con **todos los valores** (los valores pueden estar duplicados, a diferencia de las claves).
`.entrySet()`	Devuelve una **colección** con todos los pares de **clave-valor**.
`.containsKey(clave)`	Devuelve **true** si el **`HashMap`** **contiene la clave indicada** y **false** en caso contrario.

`.getKey()`	Devuelve la **clave** de un **`entrySet`** que contenga **una pareja clave-valor**. Sólo se aplica a esta, no a todo el **`HashMap`**.
	EJEMPLO
	.java
	`for(Map.Entry pareja : nombreHM.entrySet()){`
	`  System.out.println(pareja.getKey());`
	`}`

`.getValue()`	Exactamente igual que **`.getKey()`**, pero devuelve el **valor** del **`entrySet`** .

Veamos con un ejemplo el uso de varios de estos métodos:

EJEMPLO

Ejemplo.java

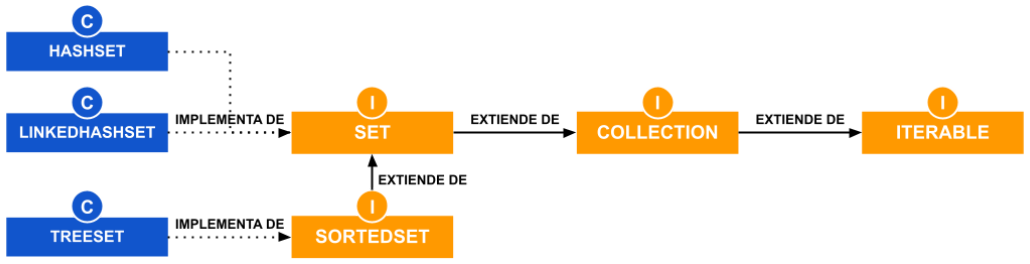
```
1  import java.util.HashMap;
2  import java.util.Map.Entry;
3
4  public class Ejemplo{
5     public static void main(String[] args){
6
7        Hashmap<Integer, String> miHashMap = new ArrayList<Integer, String>();
8
9        miHashMap.put(921, "Juan Alcántara");
10       miHashMap.put(555, "Ana Díaz");
11       miHashMap.put(212, "José Moreno");
12
13       System.out.println("HashMap accediendo por su clave:");
14       for(Integer key : miHashMap.keySet()){
15          System.out.println("Clave: "+key+", Valor: "+miHashMap.get(key));
16       }
```

```java

        System.out.println("\nHashMap accediendo por pares (clave-valor):");
        for(Entry<Integer, String> par : miHashMap.entrySet()){
            System.out.println("Clave: "+par.getKey()+", Valor: "+par.getValue();
        }

    }
}
```

Console

```
HashMap accediendo por su clave:
Clave: 921, Valor: Juan Alcántara
Clave: 555, Valor: Ana Díaz
Clave: 212, Valor: José Moreno

HashMap accediendo por pares (clave-valor):
Clave: 921, Valor: Juan Alcántara
Clave: 555, Valor: Ana Díaz
Clave: 212, Valor: José Moreno
```

5.4 - CLASES SET



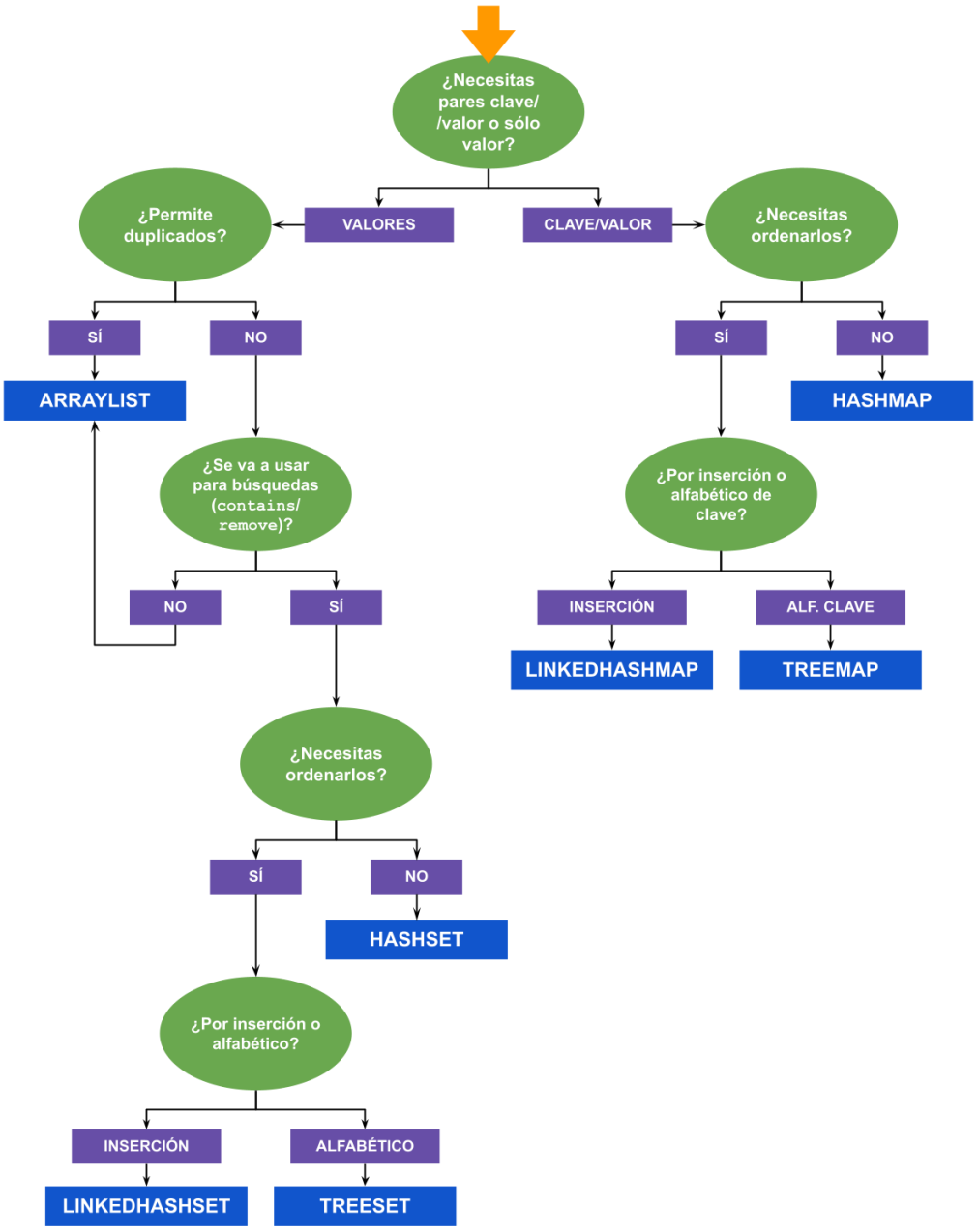
`Set` es un tipo de interfaz usado en colecciones que **no permite duplicados (a diferencia de `List`)**. Las **tres clases** que implementan de esta y sus diferencias son:

HashSet	LinkedHashSet	TreeSet (a través de SortedSet)
No mantiene el orden de inserción.	**Ordena los datos** en **orden alfabético ASCII** automáticamente.	**Mantiene** el orden de inserción.

Aunque en una clase de la interfaz `Set` no se admiten duplicados, esto nos lo podemos saltar si damos **dos objetos que aunque cambien de nombre tengan el mismo contenido**. Para evitar esto usamos la interfaz `Comparator<T>` y su método `compare()`, el cual podemos **sobreescribir** y establecer las reglas de comparación. Puedes aprender más sobre este en el **Anexo:** ***Implementación de Métodos Recomendados***.

5.5 - IMPLEMENTACIÓN DE COLECCIONES

A veces es complicado saber **cuál es la colección más óptima** para nuestro proyecto, por eso puedes usar este diagrama para orientarte y elegir la más adecuada:



U5 - **BATERÍA DE EJERCICIOS**

1	1. Crea una biblioteca de funciones (librería) para arrays (de una dimensión) de números enteros que contenga las siguientes funciones: a) generaArrayInt: Genera un array de tamaño n con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro. b) minimoArrayInt: Devuelve el mínimo del array que se pasa como parámetro. c) maximoArrayInt: Devuelve el máximo del array que se pasa como parámetro. d) mediaArrayInt: Devuelve la media del array que se pasa como parámetro. e) estaEnArrayInt: Dice si un número está o no dentro de un array. f) posicionEnArray: Busca un número en un array y devuelve la posición (el índice) en la que se encuentra. g) volteaArrayInt: Le da la vuelta a un array. Crea una clase para probar todas las funciones de la librería, además está podrá ser utilizada en cualquier otro ejercicio.
2	Crea un programa para realizar cálculos relacionados con la altura (en metros) de personas. Pedirá un valor N y luego almacenará en un array N alturas introducidas por teclado. Luego mostrará la altura media, máxima y mínima así como cuántas personas miden por encima y por debajo de la media.
3	Crea un programa que cree un array de 10 enteros y luego muestre el siguiente menú con distintas opciones: a) Mostrar valores. b) Introducir valor. c) Salir. La opción 'a' mostrará todos los valores por pantalla. La opción 'b' pedirá un valor V y una posición P, luego escribirá V en la posición P del array. El menú se repetirá indefinidamente hasta que el usuario elija la opción 'c' que terminará el programa.
4	Crea un programa que permita al usuario almacenar una secuencia aritmética en un array y luego mostrarla. Una secuencia aritmética es una serie de números que comienza por un valor inicial V, y continúa con incrementos de I. Por ejemplo, con V=1 e I=2, la secuencia sería 1, 3, 5, 7, 9... Con V=7 e I=10, la secuencia sería 7, 17, 27, 37... El programa solicitará al usuario V, I además de N (no de valores a crear).

NOTA : Utiliza los métodos de la clase 'Arrays' para ayudarte a resolver los siguientes ejercicios:

5	Crea un programa que pida al usuario dos valores N y M y luego cree un array de tamaño N que contenga M en todas sus posiciones. Luego muestra el array por pantalla.
6	Crea un programa que cree un array de números enteros e introduzca la siguiente secuencia de valores: `1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5,` etc. hasta introducir 10 diez veces, y luego la muestre por pantalla. En esta ocasión has de utilizar Arrays.fill().

7	Crea un programa que pida al usuario 20 valores enteros e introduzca los 10 primeros en un array y los 10 últimos en otro array. Por último, comparará ambos arrays y le dirá al usuario si son iguales o no.
8	Crea un programa que cree un array de tamaño 30 y lo rellene con valores aleatorios entre 0 y 9 . Luego ordena los valores del array y los mostrará por pantalla.
9	Necesitamos crear un programa para mostrar el ranking de puntuaciones de un torneo de ajedrez con 8 jugadores. Se le pedirá al usuario que introduzca las puntuaciones de todos los jugadores (habitualmente valores entre 1000 y 2800, de tipo entero) y luego muestre las puntuaciones en orden descendente (de la más alta a la más baja).
10	. Crea un programa que cree un array de tamaño 1000 y lo rellene con valores enteros aleatorios entre 0 y 99. Luego pedirá por teclado un valor N y se mostrará por pantalla si N existe en el array, además de cuantas veces.
11	Define tres arrays de 20 números enteros cada una, con nombres numero, cuadrado y cubo. Carga el array numero con valores aleatorios entre 0 y 100. En el array cuadrado se deben almacenar los cuadrados de los valores que hay en el array numero. En el array cubo se deben almacenar los cubos de los valores que hay en numero. A continuación, muestra el contenido de los tres arrays dispuesto en tres columnas.
12	Escribe un programa que lea 15 números por teclado y que los almacene en un array. Rota los elementos de ese array, es decir, el elemento de la posición 0 debe pasar a la posición 1, el de la 1 a la 2, etc. El número que se encuentra en la última posición debe pasar a la posición 0. Finalmente, muestra el contenido del array.
13	Escribe un programa que genere 100 números aleatorios del 0 al 20 y que los muestre por pantalla separados por espacios. El programa pedirá entonces por teclado dos valores y a continuación cambiará todas las ocurrencias del primer valor por el segundo en la lista generada anteriormente. Los números que se han cambiado deben aparecer entrecomillados.
14	Escribe un programa que rellene un array de 20 elementos con números enteros aleatorios comprendidos entre 0 y 400 (ambos incluidos). A continuación el programa mostrará el array y preguntará si el usuario quiere resaltar los múltiplos de 5 o los múltiplos de 7. Seguidamente se volverá a mostrar el array escribiendo los números que se quieren resaltar entre corchetes.
15	Un restaurante nos ha encargado una aplicación para colocar a los clientes en sus mesas. En una mesa se pueden sentar de 0 (mesa vacía) a 4 comensales (mesa llena). Cuando llega un cliente se le pregunta cuántos son. De momento el programa no está preparado para colocar a grupos mayores a 4, por tanto, si un cliente dice por ejemplo que son un grupo de 6, el programa dará el mensaje “Lo siento, no admitimos grupos de 6, haga grupos de 4 personas como máximo e intente de nuevo”. Para el grupo que llega, se busca siempre la primera mesa libre (con 0 personas). Si no quedan mesas libres, se busca donde haya un hueco para todo el grupo, por ejemplo si el grupo es de dos personas, se podrá colocar donde haya una o dos personas. Inicialmente, las mesas se cargan con valores aleatorios entre 0 y 4. Cada vez que se sientan nuevos clientes se debe mostrar el estado de las mesas. Los grupos no se pueden romper aunque haya huecos sueltos suficientes. El funcionamiento del programa se muestra a continuación:

```
|Mesa num:   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
|Ocupación: | 1 | 0 | 2 | 4 | 2 | 2 | 3 | 4 | 0 | 2

¿Cuántos son? (Introduzca -1 para salir del programa): 4

Por favor, siéntense en la mesa número 2.

|Mesa num:   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
|Ocupación: | 1 | 4 | 2 | 4 | 2 | 2 | 3 | 4 | 0 | 2

¿Cuántos son? (Introduzca -1 para salir del programa): 3

Por favor, siéntense en la mesa número 9.

|Mesa num:   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
|Ocupación: | 1 | 4 | 2 | 4 | 2 | 2 | 3 | 4 | 3 | 2

¿Cuántos son? (Introduzca -1 para salir del programa): 7

Lo siento, no admitimos grupos de 7, haga grupos de 4 personas como máximo e intente de nuevo.
|Mesa num:   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
|Ocupación: | 1 | 4 | 2 | 4 | 2 | 2 | 3 | 4 | 3 | 2

¿Cuántos son? (Introduzca -1 para salir del programa): 3

Tendrán que compartir mesa. Por favor, siéntense en la mesa número 1.

|Mesa num:   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
|Ocupación: | 4 | 4 | 2 | 4 | 2 | 2 | 3 | 4 | 3 | 2

¿Cuántos son? (Introduzca -1 para salir del programa): -1
Gracias por su visita.
```

16 Crea un programa que cree una matriz de tamaño 5x5 que almacene los números del 1 al 25 y luego muestre la matriz por pantalla.

17 Define un array de números enteros de 3 filas por 6 columnas con nombre num y asigna los valores según la siguiente tabla. Muestra el contenido de todos los elementos del array dispuestos en forma de tabla como se muestra a continuación.

	Columna 0	Columna 1	Columna 2	Columna 3	Columna 4	Columna 5
Fila 0	0	30	2	0	0	7
Fila 1	75	0	0	0	0	0
Fila 2	0	0	-2	9	0	11

18 Crea un programa que cree una matriz de 10x10 e introduzca los valores de las tablas de multiplicar del 1 al 10 (cada tabla en una fila). Luego mostrará la matriz por pantalla.

19 Crea un programa que cree una matriz de tamaño NxM (tamaño introducido por teclado) e introduzca en ella NxM valores (también introducidos por teclado). Luego deberá recorrer la matriz y al final mostrar por pantalla cuántos valores son mayores que cero, cuántos son menores que cero y cuántos son igual a cero.

20 Escribe un programa que pida 20 números enteros. Estos números se deben introducir en un array de 4 filas por 5 columnas. El programa mostrará las sumas parciales de filas y columnas igual que si de una hoja de cálculo se tratara. La suma total debe aparecer en la esquina inferior derecha.

					Σ fila 0
					Σ fila 1
					Σ fila 2
					Σ fila 3
Σ columna 0	Σ columna 1	Σ columna 2	Σ columna 3	Σ columna 4	TOTAL

21 Realiza un programa que rellene un array de 6 filas por 10 columnas con números enteros positivos comprendidos entre 0 y 1000 (ambos incluidos). A continuación, el programa deberá dar la posición tanto del máximo como del mínimo.

22 Necesitamos crear un programa para almacenar las notas de 4 alumnos (llamados “Alumno 1”, “Alumno 2”, etc.) y 5 asignaturas. El usuario introducirá las notas por teclado y luego el programa mostrará la nota mínima, máxima y media de cada alumno. (no es necesario guardar el nombre del alumno, sabremos que alumno es por la fila) .

23	Necesitamos crear un programa para registrar sueldos de hombres y mujeres de una empresa y detectar si existe brecha salarial entre ambos. El programa pedirá por teclado la información de N personas distintas (valor también introducido por teclado). Para cada persona, pedirá su género (0 para varón y 1 para mujer) y su sueldo. Esta información debe guardarse en una única matriz. Luego se mostrará por pantalla el sueldo medio de cada género.
24	Crea un programa que pida una cadena de texto por teclado y luego muestre cada palabra de la cadena en una línea distinta.
25	Crea un programa que pida dos cadenas de texto por teclado y luego indique si son iguales, además de si son iguales sin diferenciar entre mayúsculas y minúsculas.
26	Crea un programa que pida por teclado tres cadenas de texto: nombre y dos apellidos. Luego mostrará un código de usuario (en mayúsculas) formado por la concatenación de las tres primeras letras de cada uno de ellos. Por ejemplo si se introduce “Juan”, “Moreno” y “Carmona” mostrará “JUAMORCAR”.
27	Crea un programa que muestre por pantalla cuantas vocales de cada tipo hay (cuantas ‘a’, cuantas ‘e’, etc.) en una frase introducida por teclado. No se debe diferenciar entre mayúsculas y minúsculas. Por ejemplo dada la frase “Mi mama me mima” dirá que hay: • Nº de A's: 3 • Nº de E's: 1 • Nº de I's: 2 • Nº de O's: 0 • Nº de U's: 0
28	Crea un programa que cuente las vocales y consonantes de una frase. Ten cuidado con los espacios, se deben ignorar.
29	Crear un programa que dada una cadena invertir la misma y mostrar por pantalla. Ejemplo: Entrada: "casa blanca" Salida: "acnalb asac"
30	Crear un programa que dada una cadena, y un carácter, indique cuántas veces se repite el carácter en la cadena
31	Crear un programa que lea una frase y encuentre la palabra de mayor longitud. El programa debe imprimir tanto la palabra como el número de caracteres de la misma.
32	Realiza un programa que lea una frase por teclado e indique si la frase es un palíndromo o no (ignorando espacios y sin diferenciar entre mayúsculas y minúsculas). Supondremos que el usuario solo introducirá letras y espacios (ni comas, ni puntos, ni acentos, etc.). Un palíndromo es un texto que se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo: *- Arriba la birra* *- Allí va Ramón y no maravilla* *- Ana lava lana* *- Ella te da detalle*
33	Crear un programa que multiplique dos matrices 4x4.

34	Programa que genera una matriz representando un triángulo invertido alineado a la izquierda realizado con asteriscos, cuyo tamaño será especificado por el usuario. La matriz será mostrada en pantalla finalmente. Por ejemplo, si se indica el tamaño 5 deberá aparecer:
	`*****`
	`****`
	`***`
	`**`
	`*`

35	Ejercicio sala de cine: • Crear una clase enumerada Reserva con los valores libre y ocupado.

• Crear un clase Asiento con las el atributo enumerado y otros que consideres oportunos, una numeración, hora, etc.

• Crear la clase principal e introducir una matriz de 3x2 que se llame sala y poblarla con 6 asientos, todos ellos por defecto a libre.

• Simular con un bucle (do-while) la petición de una fila y una columna del cine y comprobar si esta libre para dejarla ocupada por el espectador hasta que introduzca por ejemplo false y no quiera pedir más entradas o hasta que el cine se complete (este todo ocupado).

• Mostrar para finalizar como se quedo la sesión de esa sala.

Nota: Se puede crear un menú que permita ver la ocupación actual y reservar asiento.

>> EJERCICIOS DE RECURSIVIDAD

1	Sumar los números naturales hasta N (se lo damos nosotros) de forma recursiva.
2	Recorrer un array cualquiera de forma recursiva.
3	Calcular el valor de la posición fibonacci X usando recursividad.