5. TEMA 5: Almacenamiento de la información en estructuras de datos.

En este tema veremos como podemos almacenar diferentes tipos de datos y objetos en estructuras que nos permiten acceder a ellos a través de una sintaxis o a través de una interfaz de acceso a sus elementos.

Comprenderemos el funcionamiento de estas estructuras. Como se organizan y como se usan.

Gradualmente irá aumentando la complejidad de las mismas, desde las estructuras estáticas, hasta la dinámicas.

Conoceremos el funcionamiento de los tipos abstractos de datos y aprenderemos a usar los objetos que los implementan.

5.1 Arrays unidimensionales.

El concepto de array consiste en reunir un conjunto de valores de un mismo tipo a través del acceso de un único nombre de variable, de tal manera que mediante una sintaxis concreta podemos acceder al elemento deseado. Cuando nos referimos a arrays unidimensionales hacemos referencia a que sólo tiene una dimensión, pero dentro de esa dimensión podemos introducir los elementos que deseemos, generalmente a los arrays unidimensionales se les suele llamar “vector”. Una característica que tienen los arrays que aquí veremos es que son estáticos. Esto quiere decir que previamente debemos saber cuantos elementos va a contener el array, una vez hemos creado el array no podemos agregar más elementos si hemos llegado al máximo.

Una característica principal en los arrays es el uso de los corchetes “[ ]”. La sintaxis para definir un array de enteros por ejemplo sería de la siguiente manera:

```
int miArray[];
```

En determinadas bibliografías también podemos encontrarlo como:

```
int[] miArray;
```

Ahora mismo sólo hemos definido el array, pero aún no hemos reservado su espacio en memoria ni hemos establecido cuantos elementos va a contener, para hacerlo podemos hacerlo en la misma línea de la definición de la siguiente manera:

```
int miArray[]=new int[5];
```

Así lo que tenemos es un conjunto de 5 elementos enteros. Al inicializarlo lo que tenemos son 5 celdas en memoria de 5 enteros, que inicialmente tendrán el valor por defecto 0. Gráficamente en memoria podría representarse así:

0	0	0	0	0

Otra manera de definir el array e iniciarlo con unos valores determinados sería así:

```
int miArray[]={1,5,2,7,-2};
```

Gráficamente sería:

1	5	2	7	-2

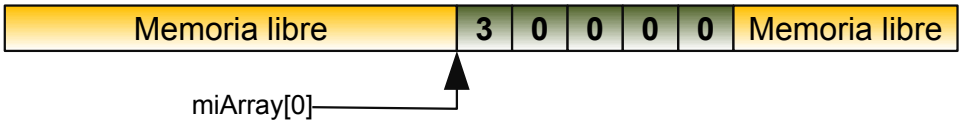
Si lo que quiero es inicializar el array en una línea diferente a la de la definición se hace de la siguiente manera:

```
int miArray[];
miArray=new int[5];
```

Ahora veremos como accedemos a cada uno de los elementos del array. Si lo que quiero es introducir un valor en la primera celda del array se hace de la siguiente manera:

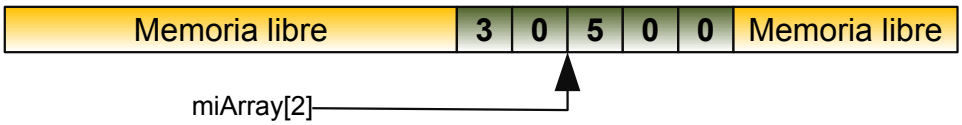
```
int miArray[]=new int[5];
miArray[0]=3;
```

Reflejemos como afecta esto en memoria:



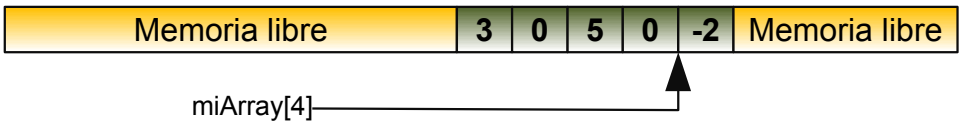
Ahora vamos a introducir el valor 5 en la tercera celda de memoria:

```
miArray[2]=5;
```



Si queremos introducir el número -2 en la última posición:

```
miArray[4]=-2;
```



Si lo que quiero es por ejemplo mostrar por pantalla el contenido de lo que hay en la tercera celda:

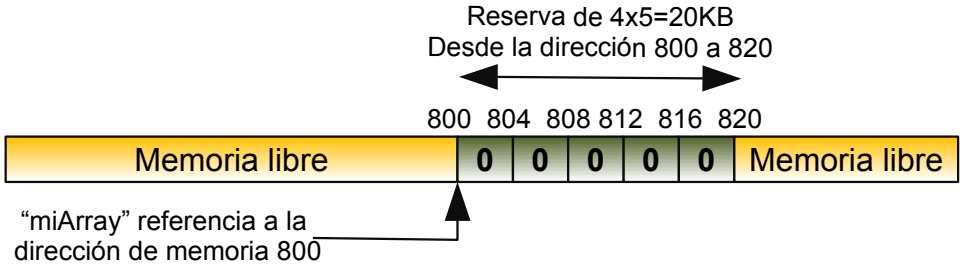
```
System.out.println("El valor de la tercera celda es:"+miArray[2]);
```

Como podemos ver cuando declaro el array he puesto que va a ser de 5 elementos, pero las posiciones de estos elementos van desde 0 a 4. Entonces, ¿porqué empieza en 0 y termina en 4?. La explicación es bien sencilla, esto es porque un array es un elemento especial en java, el cual referencia una dirección de memoria, la cual va a ser así mismo la referencia para ir accediendo a cada uno de los elementos, el índice que pasamos entre corchetes es lo que nos ayuda a calcular la siguiente dirección de memoria a acceder. Cuando defino por ejemplo un array de 5 enteros, al declararlo lo que estamos haciendo es reservar ese espacio en memoria para el array, teniendo en cuenta que cada entero ocupa un espacio de 4 Kbytes, para 5 elementos el espacio reservado sería de 20 Kbytes.Para poder entender esto lo mejor es ilustrar todo el el proceso desde que se crea

el array y como se accede a los elementos.

```
int miArray[]=new int[5];
```

Reflejemos como afecta esto en memoria:



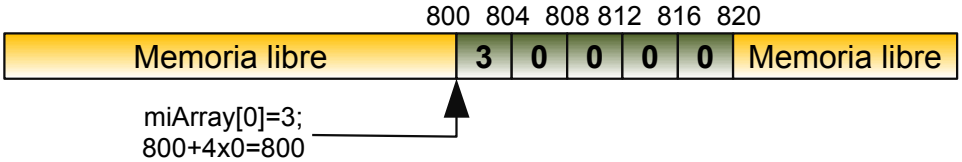
Ahora vamos a introducir el valor 3 en la primera celda de memoria:

```
miArray[0]=3;
```

Sabiendo que miArray referencia a la dirección de memoria 800, si calculamos la posición con el índice sería:

800+(tamaño del tipo)xíndice → 800+4x0=800

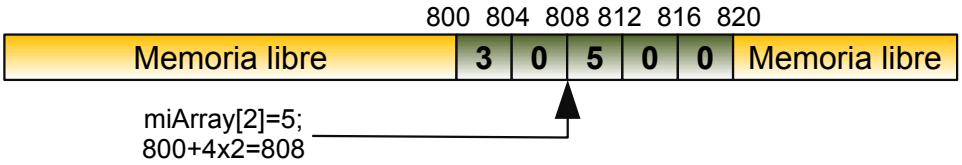
Por tanto sabemos que el valor 3 que queremos escribir lo debemos escribir en los 4Bytes consecutivos a la dirección que hemos hallado que es la 800. Por tanto desde la dirección 800 a la 804 va a estar ocupado por el valor 3:



Ahora vamos a introducir el valor 5 en la tercera celda de memoria:

```
miArray[2]=5;
```

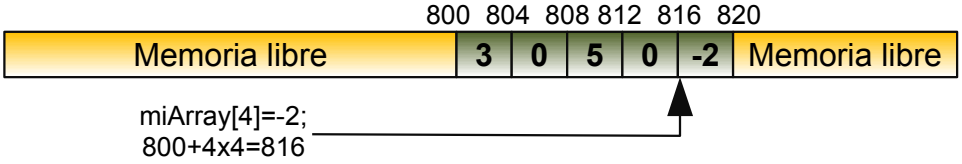
Por tanto desde la dirección 808 hasta la 812 va a estar ocupada por el valor 5:



Si queremos introducir el número -2 en la última posición:

```
miArray[4]=-2;
```

Por tanto desde la dirección 816 hasta la 820 va a estar ocupada por el valor -2:

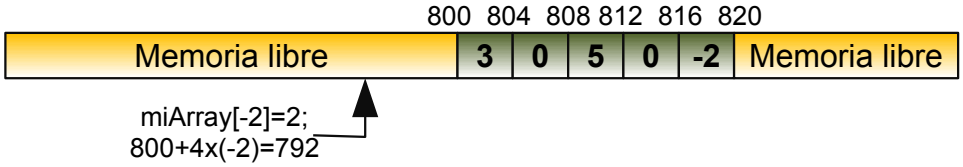


Ahora, ¿que sucede si intento introducir un índice negativo o que se encuentre fuera del array?, pues en ese caso recibiremos un error indicando que nos hemos salido de los

límites de array, ya que esto provoca un error al acceder a la memoria. Lógicamente si yo he reservado la dirección desde la 800 hasta la 820 no puedo acceder a otras direcciones de memoria que no me pertenecen. Si intento acceder con un índice negativo:

```
miArray[-2]=2;
```

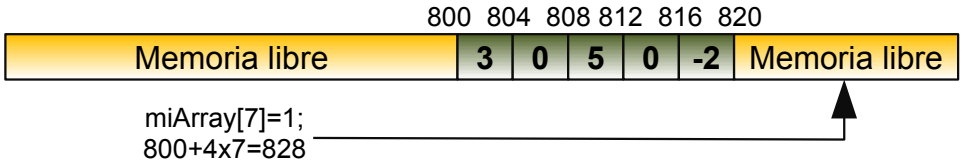
Si realizamos el cáculo: 800+4x(-2)=792. Esa dirección de memoria no me pertenece.



Ahora veamos con un número fuera del array:

```
miArray[7]=1;
```

Si realizamos el cáculo: 800+4x7=828. Esa dirección de memoria no me pertenece.



Como vemos ahora es cuando cobra sentido el concepto mencionado anteriormente en el cual se dice que los arrays son estáticos, una vez he definido la cantidad de elementos, esta no puede ser aumentada.

Una vez que ya sabemos como funcionan los arrays la cuestión que debemos plantearnos es, ¿para qué sirve esto realmente?. Imaginemos que quiero guardar la nota de 5 alumnos en mi programa para poder usarlas después. Anteriormente sin conocer el uso de los arrays tendríamos que haber creado 5 variables:

```
notaAlumno1=2;
notaAlumno2=5;
notaAlumno3=7;
notaAlumno4=6;
notaAlumno5=9;
```

Además para ir mostrándolas tendría que hacer lo siguiente:

```
System.out.println(notaAlumno1);
System.out.println(notaAlumno2);
System.out.println(notaAlumno3);
System.out.println(notaAlumno4);
System.out.println(notaAlumno5);
```

Esto con arrays resulta mucho mas sencillo, ya que los arrays se recorren y solo cambiamos el valor del índice. Para recorrer un array la estructura más recomendable es un “for”. Veamos lo anterior como sería con arrays:

```
public static void main(String[] args) {
      int posicion;
      int notas[]={2,5,7,6,9};
      for(posicion=0;posicion<5;posicion++){
         System.out.println("Nota alumno "+posicion+":"+notas[posicion]);
      }
}
```

El resultado sería:

```
Nota alumno 0:2
Nota alumno 1:5
Nota alumno 2:7
Nota alumno 3:6
Nota alumno 4:9
```

El código es mucho más sencillo y compacto. Como se puede observar, el "for" tiene una variable "posicion" que hace referencia al índice del array, y esta variable va a tomar los valores desde 0 hasta 4. En el "for" hemos puesto que la condición es "posicion<5", porque sabemos que nuestro array consta de 5 elementos.

Tal y como se ha dicho, un array es un elemento especial en Java. Una de las características que lo hace especial, a parte de su sintaxis, es que comparte las características de los objetos, por tanto tiene los métodos que provienen de "Object" y además cuando igualamos un array a otro lo que estamos haciendo es copiar la referencia a memoria, pero las modificaciones sobre los elementos son para ambos las mismas. Por ejemplo, si tenemos:

```
public static void main(String[] args) {
      int posicion;
      int notas[]={2,5,7,6,9};
      int notasCopia[];
      notasCopia=notas;
      notasCopia[2]=-10;
      for(posicion=0;posicion<5;posicion++){
         System.out.println("Nota alumno "+posicion+":"+notas[posicion]);
      }
}
```

El resultado es:

```
Nota alumno 0:2
Nota alumno 1:5
Nota alumno 2:-10
Nota alumno 3:6
Nota alumno 4:9
```

Esto es así porque aunque inicialmente "notas" en la posición 2 antes tenía el valor 7, al realizar la copia con "notasCopia" y modificar el valor de esa posición por el valor -10, cuando mostramos los valores del array "notas" vemos que ha sido modificado. Entonces, ¿que ocurre si lo que yo quiero es tener una copia del array que sea independiente?, lo que debemos hacer es usar el método "clone()" proporcionado por "Object" el cual nos permite clonar un objeto exáctamente igual en la memoria. Por tanto:

```
public static void main(String[] args) {
      int posicion;
      int notas[]={2,5,7,6,9};
      int notasCopia[];
      notasCopia=notas.clone();
      notasCopia[2]=-10;
      for(posicion=0;posicion<5;posicion++){
         System.out.println("Nota alumno "+posicion+":"+notas[posicion]);
      }
}
```

Ahora el resultado es:

```
Nota alumno 0:2
Nota alumno 1:5
Nota alumno 2:7
Nota alumno 3:6
Nota alumno 4:9
```

Ejercicio Auto-evaluación A5.1: *Realizar una función que rellene un array con los 100 primeros números enteros y los muestre por pantalla en orden ascendente.*

Ejercicio Auto-evaluación A5.2: *Realizar otra función que haga lo mismo que la anterior solo que esta vez los muestra por orden descendente.*

5.1.1 Arrays y métodos.

Lo explicado anteriormente es tambien válido cuando usámos métodos, si le paso un array a un método y lo modifico dentro del mismo, cuando acabe el método las modificaciones en el array permanecen. Pero, ¿como le paso un array a un método?, la sintaxis es simple, solo hay que definir un array de la misma dimensión en los parámetros. Imaginemos que queremos mostrar las notas de los alumnos a través de un método, sería de la siguiente manera:

```
public class Tema5ejemplo2 {
      public static void mostrarNotas(int n[],int tam){
            int indice;
            for(indice=0;indice<tam;indice++){
                  System.out.println("Nota alumno "+indice+":"+n[indice]);
            }
      }

      public static void main(String[] args) {
            int notas[]={2,5,7,6,9};

            mostrarNotas(notas,5);
      }
}
```

Si nos fijamos, en la cabecera del método indico que recibe un array de una dimensión, por eso solo tiene unos corchetes, y además también le paso el tamaño del array para poder recorrerlo. En la segunda parte que está marcada se puede ver que cuando le paso el array "notas", no le pongo los corchetes, no son necesarios debido a que en la cabecera de la función ya sabemos que recibimos un array, y que lo que lógicamente pasamos es la referencia al array.

Antes se ha comentado que un array tiene unas características especiales, y que tiene ciertos métodos y atributos. Entre los atributos encontramos uno muy valioso que es el "length", este atributo nos sirve para saber que tamaño tiene el array, por tanto podríamos desechar el segundo parámetro del método "mostrarNotas" ya que no es necesario pasarle el tamaño, podemos averiguarlo a través del mismo array que nos pasan como parámetro, de tal manera que la función nos quedaría de la siguiente manera:

```
public static void mostrarNotas(int n[]){
      int indice;
      for(indice=0;indice<n.length;indice++){
            System.out.println("Nota alumno "+indice+":"+n[indice]);
      }
}
```

En el caso mostrado "n.length" vale 5.

Y la llamada desde el main se quedaría así:

```
      mostrarNotas(notas);
```

Como hemos dicho antes, lo que paso al método es la referencia, entonces, ¿qué ocurre si tengo un método "incrementaNotas" que les suma un 1 a cada una de las notas?, lógicamente esos cambios se quedan permanentes. Veamos un ejemplo:

```
public class Tema5ejemplo2 {
      public static void incrementaNotas(int n[]){
            int i;
            for(i=0;i<n.length;i++){
                  n[i]=n[i]+1;
            }
      }
      public static void mostrarNotas(int n[]){
            int indice;
            for(indice=0;indice<n.length;indice++){
                  System.out.println("Nota alumno "+indice+":"+n[indice]);
            }
      }

      public static void main(String[] args) {
            int notas[]={2,5,7,6,9};
            incrementaNotas(notas);
            mostrarNotas(notas);
      }
}
```

El resultado es:

```
Nota alumno 0:3
Nota alumno 1:6
Nota alumno 2:8
Nota alumno 3:7
Nota alumno 4:10
```

Ejercicio Auto-evaluación A5.3: *¿Cómo harías para que en el ejemplo anterior al pasarle el array a la función "incrementaNotas" los cambios no queden reflejados en el array original?.*

Ahora solo nos queda saber como se puede hacer que un método nos devuelva un array, vamos a imaginarnos que lo queremos es tener el array de "notas" inicial y luego otro array con las "notasIncrementadas", para ello la función debe devolver un array con las notas modificadas. Por tanto sería así:

```
public class prueba {
	public static int[] incrementaNotas(int n[]){
		int modificado[]=new int[n.length];
		int i;
		for(i=0;i<n.length;i++){
			modificado[i]=n[i]+1;
		}
		return modificado;
	}
	public static void mostrarNotas(int n[]){
		int indice;
		for(indice=0;indice<n.length;indice++){
			System.out.println("Nota alumno "+indice+":"+n[indice]);
		}
	}

	public static void main(String[] args) {
		int notas[]={2,5,7,6,9};
		int notasIncrementadas[];
		notasIncrementadas=incrementaNotas(notas);
		mostrarNotas(notas);
		mostrarNotas(notasIncrementadas);
	}
}
```

Como siempre están resaltadas las partes mas relevantes. El primer marcado hace referencia a lo que devuelve la función, esta vez no es un "void" sino que devuelve un array de enteros de una dimensión "int[]". Lo segundo que hay marcado es para observar que creamos un nuevo array llamado "modificado" que tiene la misma longitud que el array que nos pasan como parámetro. El cuarto marcado es para resaltar que en modificado vamos guardando los nuevos valores del resultado del incremento. El quinto marcado es para resaltar que como nuestra función devuelve un array de una dimensión tenemos que hacer un "return modificado" para devolver el array modificado. Por último

en el último marcado vemos que creo un array llamado “notasIncrementadas” el cual en la segunda línea va a ser igual al resultado del método “incrementarNotas”, por tanto igual que el array “modificado” que devuelve este.

5.1.2 Arrays de objetos.

Lo explicado anteriormente no sólo es válido para los tipos básicos como “int”, sino que también lo es para los objetos, de tal manera que yo puedo tener un array de objetos del tipo que yo desee. Entonces puedo crear perfectamente por ejemplo un array de objetos persona. Imaginemos que tenemos la siguiente clase “Persona.java”:

```
public class Persona {
	private String nombre;
	private int edad;
	public Persona(String n,int e){
		nombre=n;
		edad=e;
	}
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}

	public void mostrar(){
		System.out.println("Nombre:"+nombre+"\tedad:"+edad);
	}
}
```

Yo puedo crear por ejemplo en el “main” un array con 4 personas de esta manera:

```
public class personaArrayMain {
	public static void main(String[] args) {
		Persona gente[]=new Persona[4];
		gente[0]=new Persona("Pepe",29);
		gente[1]=new Persona("María",23);
		gente[2]=new Persona("Juan",20);
		gente[3]=new Persona("Ana",38);
		int i;

		for(i=0;i<gente.length;i++){
			gente[i].mostrar();
		}
	}
```

```
}
```

En el primer marcado se puede ver como se define un array “gente” de 4 objetos del tipo “Persona”. En el segundo marcado vemos que para cada una de las posiciones creo una persona diferente. En el último marcado se puede ver que al acceder a una de las personas, cuando ponemos el punto “.” podemos acceder a los atributos y métodos de esa persona concreta, en concreto al método mostrar de cada una de las personas, que nos muestra por pantalla los datos de la persona.

El resultado al ejecutar el “main” será:

```
Nombre:Pepe     edad:29
Nombre:María    edad:23
Nombre:Juan     edad:20
Nombre:Ana      edad:38
```

Ejercicio Auto-evaluación A5.4: *Realizar una función que nos muestre solo el primer y último valor de un array de 8 elementos. Por ejemplo, si tenemos un array como:*

10	2	3	-5	8	1	50	-3

Debería mostrar por pantalla:
Primer elemento:10
Último elemento:-3

Ejercicio Auto-evaluación A5.5: *Realizar una función a la que se le pasa un array de X elementos (esto quiere decir que la longitud es la que queramos definir) y un rango de números, y nos muestra todos los números del array que se encuentran entre ese rango separados por coma. Ejemplo:*

1	7	3	10	0	-2	-9	2	11	30	24	1	4

Y el rango es -3,3. Por tanto mostraremos todos los números que se encuentren entre -3 y 3:
1,3,0,-2,2,1,

Ejercicio Auto-evaluación A5.6: *Realizar una función que se le pasan dos arrays de tamaño 7 y en un tercer array guarda el mayor de los números de esa fila. Usar los números que se quiera o los del ejemplo, queda a elección del alumno. Ejemplo:*

array1	array2		array3
1	20	el mayor es →	20
4	2	el mayor es →	4
3	30	el mayor es →	30
7	34	el mayor es →	34
10	0	el mayor es →	10
9	-10	el mayor es →	9
40	4	el mayor es →	40

Ejercicio Auto-evaluación A5.7: *Realizar una función a la que se le pasan dos palabras String y nos devuelve un array con cada uno de los caracteres de ambos String concatenados y separados por un espácio en blanco. Ejemplo:*

***palabra1**: "hola" y **palabra2**: "adios" **resultado**→*

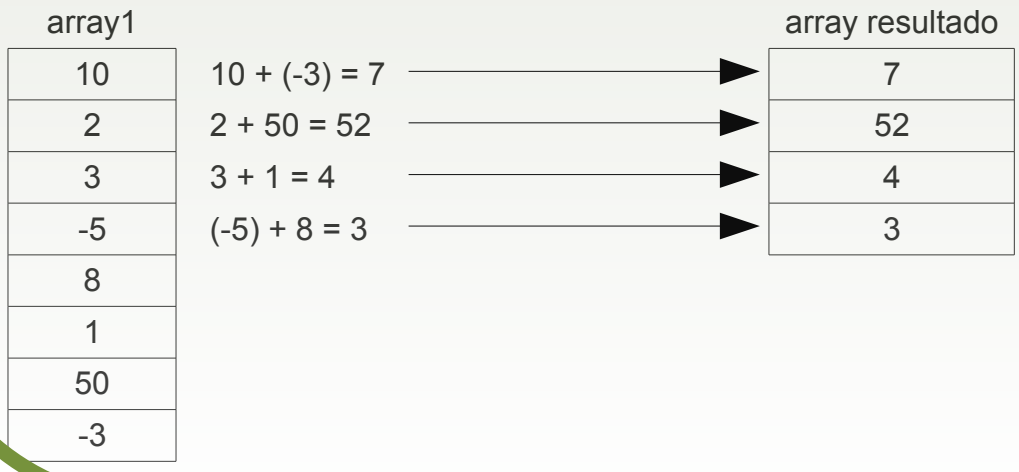
h	o	l	a		a	d	i	o	s

Ejercicio Auto-evaluación A5.8: *Realizar una función que hace justo lo contrario a la anterior. Se le pasa un array que contiene letras que están separadas por un espacio y nos muestra por pantalla las palabras separadas. Ejemplo:*

e	s	t	o		e	s		u	n		e	j	e	m	p	l	o

El resultado de ese array sería esto:
esto
es
un
ejemplo

Ejercicio Auto-evaluación A5.9: *Realizar una función que a partir de un array que le pasamos de X elementos (puede ser cualquiera definido por el usuario) nos calcula un nuevo array sumando el primer y último número y poniendo la suma en la primera celda, después el segundo y penúltimo y poniendo el resultado en la segunda celda, y así sucesivamente. Ejemplo:*



5.1.3 Ordenación y búsqueda en arrays.

Una vez conocemos la funcionalidad y uso de los arrays debemos también aprender a

ordenarlos, ya que la inserción de elementos no siempre será ordenada, de tal manera que disponemos de multitud de algoritmos de ordenación que nos ayudarán a facilitar esta tarea. Concretamente veremos dos de ellos, la ordenación por selección o “Sort” y la ordenación por el algoritmo de la “Burbuja”.

La ordenación en arrays tiene una clara funcionalidad, y es que a la hora de buscar un elemento, si el array está ordenado la búsqueda siempre puede ser más rápida. Así que también veremos dos métodos de búsqueda, el primero de ellos es la búsqueda lineal, en la cual no importa si el array está ordenado, busca elemento a elemento, lo cual puede ser muy costoso, y el segundo que veremos será la búsqueda binaria, que para funcionar correctamente necesita que el array esté ordenado, y nos asegura una mayor eficiencia que la búsqueda lineal.

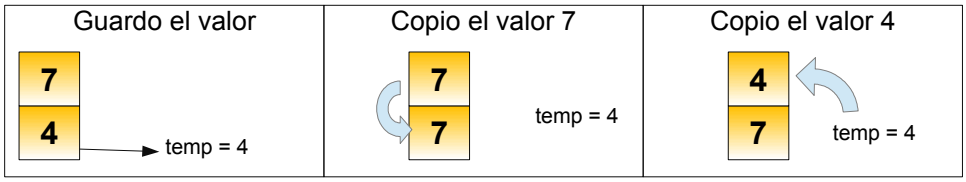
5.1.3.1 Ordenación por selección (Sort).

La ordenación por selección tiene una premisa muy sencilla que consiste en recorrer una vez el array y encontrar el elemento más pequeño suponiendo como punto de partida que el más pequeño es el que tenemos en la primera posición, una vez lo hemos recorrido todo y tenemos el más pequeño estamos seguros de que es el más pequeño en todo el array, por tanto lo colocamos en la primera posición, y para ello hacemos un intercambio, el elemento que estaba en la primera posición lo intercambiamos por la posición del elemento más pequeño. A continuación volvemos a hacer lo mismo, sólo que esta vez en la segunda iteración empezamos a recorrer a partir del segundo elemento, porque el primer elemento del array estamos seguros de que es el más pequeño, una vez encontrado de nuevo el más pequeño lo colocamos en la segunda posición, volvemos a hacer el intercambio de nuevo, en la tercera iteración empezaríamos a partir de la tercera posición y así sucesivamente. Cuando hemos recorrido el array tantas veces como elementos tiene nos hemos asegurado de que todos están ordenados.

La desventaja de este algoritmo es que para arrays muy grandes la ordenación puede ser muy lenta. Hay que tener en cuenta que siempre recorreremos tantas veces los elementos que hay sin ordenar como elementos tenga el array.

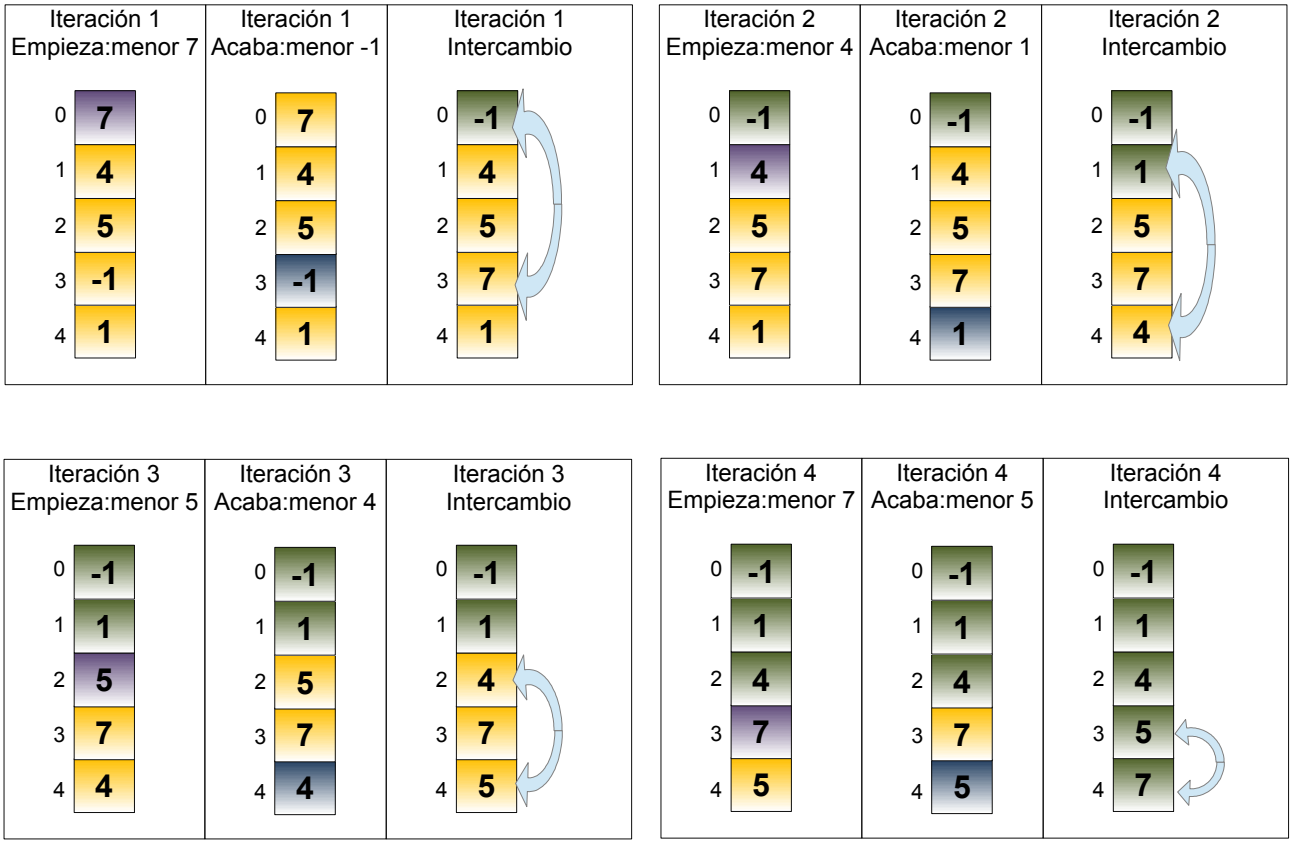
Una función muy importante es la que llamaremos de intercambio, que lo único que hace es intercambiar dos elementos en un array. Esta función es importante saber como funciona, porque cuando voy a intercambiar dos elementos tengo que guardar al menos uno de ellos en una tercera variable para hacer la sustitución. Pongamos el siguiente ejemplo tengo estos dos elementos y quiero intercambiarlos:

Si lo que quiero es intercambiarlos necesito antes de copiar el 7 en el puesto del número 4 sacar este número, el 4 lo saco a una variable que puedo llamar temporal y procedo con el intercambio.



Veamos de manera gráfica como se resolvería el array con los números {7,4,5,-1,1}, se refleja más abajo, y lo que tenemos a la izquierda de cada casilla es la posición de la celda ya que si el array es de 5 elementos las respectivas posiciones son 0,1,2,3 y 4, el más pequeño con el que empezamos la iteración lo marcaremos siempre de color

morado, y el más pequeño encontrado después de realizar la iteración lo marcaremos de color azul, y los elementos que ya estén ordenados de color verde:



Como vemos en la última iteración todos los elementos quedan ordenados, por tanto no es necesario hacer una nueva iteración para comprobar que están ordenados, así que realizaremos tantas iteraciones como elementos tenga el array menos uno, este array tenía 5 elementos, entonces hemos realizado 5-1=4 iteraciones. Si nos fijamos en este caso en cada una de las iteraciones hemos necesitado hacer un intercambio, pero ¿que hubiese sucedido si en una de ellas el elemento que encontramos primero (el marcado en morado) hubiese sido el menor?, sencillamente hubiesemos realizado también el intercambio solo que esta vez es el mismo número.

Una vez que hemos visto el funcionamiento de manera visual, vamos a observar como quedaría el algoritmo. Primero el más básico el método de intercambio en lenguaje java:

```
public static void intercambio(int array[],int pos1,int pos2){
      int temp=array[pos1];
      array[pos1]=array[pos2];
      array[pos2]=temp;
}
```

Como se observa simplemente le pasamos el array y las dos posiciones que queremos intercambiar, usa una variable “temp” para guardar temporalmente el valor de la posición 1 y cuando machamos el valor de la posición 1 con el valor de la posición 2 podemos meter en la posición 2 el valor que tenía la posición 1 porque lo habíamos guardado en el temporal.

Ahora veamos como sería el algoritmo de ordenación en un lenguaje más parecido al

pseudocódigo (vamos a usar otra nomenclatura pero igual de legible que la que usabamos en el primer tema, así nos podemos habituar a leer pseudocódigo de diferentes maneras), "n" va a ser la longitud del array:

```
para i=1 hasta n-1
     minimo = i;
     para j=i+1 hasta n
          si array[j] < array[minimo] entonces
               minimo = j;
          fin si
     fin para
     intercambio(array,i,minimo);
fin para
```

Ejercicio Auto-evaluación A5.10: *Realizar el algoritmo de ordenación "Sort" en Java y probarlo con el array de números puestos en el ejemplo: 7,4,5,-1,1. Cuando el algoritmo acaba la ordenación debe indicarnos la cantidad de iteraciones realizadas para resolverlo.*

Ordenación por selección mejorado (Sort mejorado)

Este simplemente consiste en una pequeña modificación en el algoritmo anterior, y se basa en aprovechar al mismo tiempo que hacemos una iteración para averiguar el menor lo hacemos también para averiguar el mayor, de tal manera que en la misma iteración nos aseguramos de que el menor y el mayor están bien colocados, esto implica un menor número de iteraciones ya que son justo la mitad. Sigue siendo un algoritmo lento para arrays muy grandes.

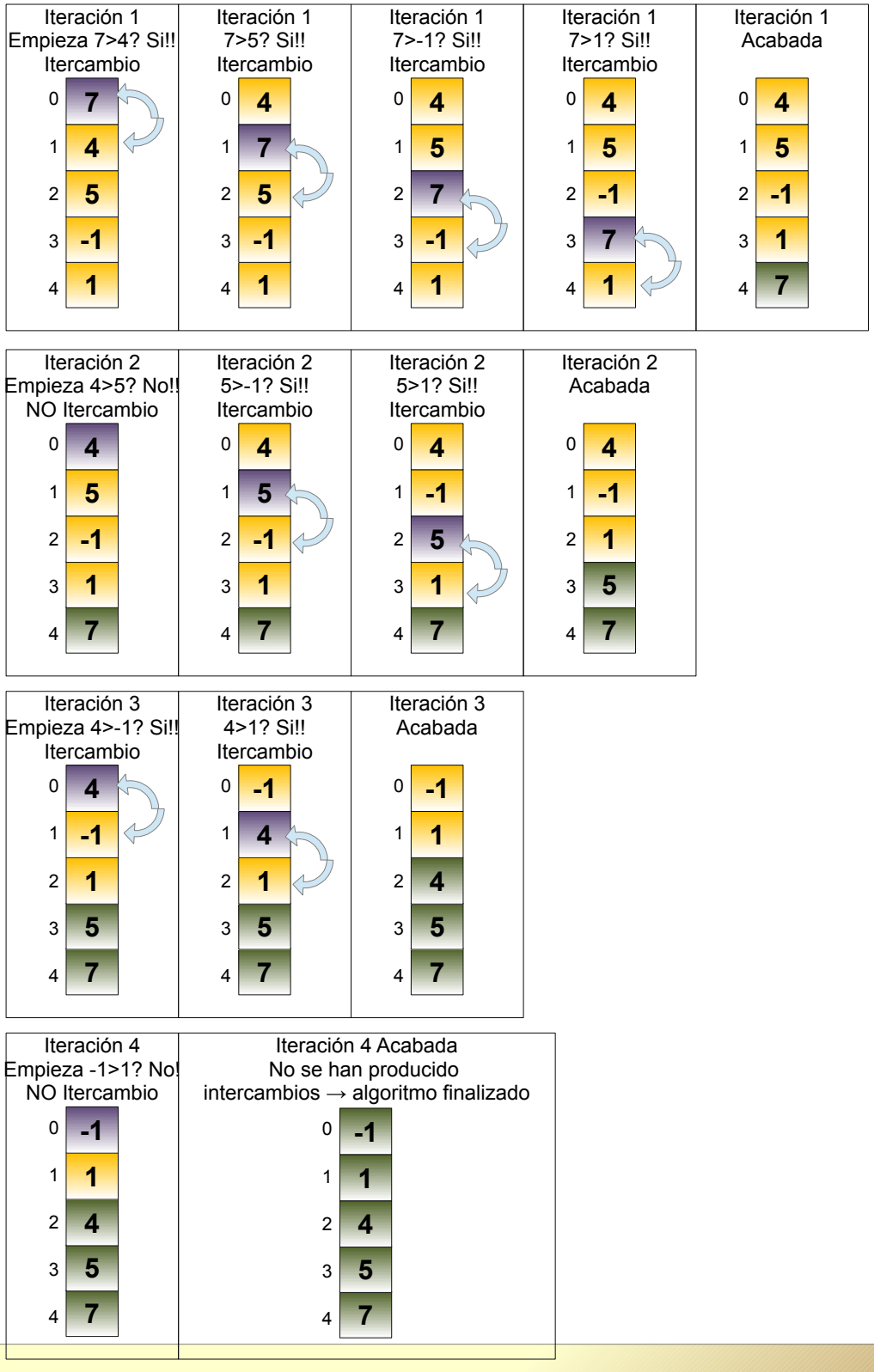
Ejercicio Auto-evaluación A5.11: *Realizar el algoritmo de ordenación "Sort" mejorado en Java y probarlo con el array de números puestos en el ejemplo: 7,4,5,-1,1. Cuando el algoritmo acaba la ordenación debe indicarnos la cantidad de iteraciones realizadas para resolverlo. Observar que son la mitad que el anterior.*

5.1.3.2 Ordenación por el método de la burbuja.

El algoritmo de ordenación de la burbuja está basado en una premisa similar a los anteriores, pero la diferencia es que conforme recorro el array si encuentro que el elemento siguiente es menor que el que estoy observando realizo el intercambio. Esto produce una repercusión clara, y es que el resultado de esto es que vamos arrastrando el elemento mayor hacia el final del array y nos aseguramos en cada pasada del array los elementos se van acercando más a su posición correcta. Este algoritmo se llama "burbuja" porque si observasemos la evolución de los elementos sobre una gráfica la sensación que produce es la de burbujar que van subiendo y se van colocando en su posición correcta. Entonces, ¿cuando acaba este algoritmo?, pues a diferencia de los anteriores este no tiene una duración fija, este algoritmo acaba cuando hemos realizado una pasada completa y no hemos realizado ningún intercambio, si esto sucede quiere decir que está totalmente ordenado, así que para ello se suele usar una variable de tipo "boolean" que nos indica cuando ha acabado.

Este algoritmo suele funcionar bien con arrays pequeños, por ejemplo de 100 elementos, pero si usamos este algoritmo con arrays grandes puede que incluso el coste sea mayor que el "Sort" debido a que el coste computacional de intercambiar dos elementos es mucho más costoso que el de simplemente leer una posición del array, por tanto si tenemos en cuenta el "Sort" por cada iteración solo hace un intercambio, en cambio la "burbuja" puede realizar cientos de intercambios en un array grande en cada iteración.

Veamos de manera gráfica como funciona este algoritmo:



Ejercicio Auto-evaluación A5.12: *Realizar el algoritmo de ordenación "burbuja" en Java y probarlo con el array de números puestos en el ejemplo: 7,4,5,-1,1. Cuando el algoritmo acaba la ordenación debe indicarnos la cantidad de iteraciones realizadas para resolverlo. Observar que son variables.*

5.1.3.3 Búsqueda lineal.

La búsqueda lineal consiste en un sencillo algoritmo que nos permite buscar un elemento dentro de un array. El único problema que tiene la búsqueda lineal es que si el array es muy grande y el elemento que queremos encontrar se encuentra casi al final, o no se encuentra, tiene que recorrer todo el array, ya que empieza a observar desde el primer elemento hasta el último en ese orden.

Veamos el algoritmo:

```
Datos de entrada al método:
  array: array en el que se desea buscar el dato
  dato: elemento que se quiere buscar.

Algoritmo: (Devuelve la posición si lo encuentra o -1 sino)
pos = 0;
tam = array.length; // Tamaño del array
Mientras (pos < tam) hacer
      Si vec[pos] == dato
            devolver pos;
      Sino
            pos = pos + 1;
      Fin Si
Fin Mientras
Devolver -1;
```

Ejercicio Auto-evaluación A5.13: *Realizar el algoritmo de búsqueda lineal en Java y probarlo con el array de números:*
{-30,-19,-21,-16,27,14,-6,8,20,21,-6,-19,20,-23,22,-13,25,24,9,-27}
Buscar el número -21, y después el 9. Observar los resultados obtenidos.
Cuando el algoritmo acaba la ordenación debe indicarnos la cantidad de elementos visitados hasta que encuentra el que buscamos.

5.1.3.4 Búsqueda binaria.

La búsqueda binaria tiene como única restricción para que funcione que el algoritmo esté ordenado, de lo contrario no funciona. La ventaja es que podemos aprovechar la información de la que disponemos previamente, que los elementos están ordenados.

Cuando búsco un elemento consiste en mirar en la mitad del array, y como está

ordenado miro si el elemento que busco es mayor o es menor que el elemento central del array, si es menor que el central miro en la mitad izquierda de nuevo, y si es mayor miro en la mitad derecha. Este proceso debe repetirse hasta que encontramos el elemento buscado o hasta que no podemos dividir mas el array.

En la mayoría de los casos este algoritmo encontrará en un número mucho menor de pasos el elemento buscado antes que el lineal.

El algoritmo sería el siguiente:

```
Datos de entrada:
  array: array en el que se desea buscar el dato
  dato: elemento que se quiere buscar.

Variables
  centro: subíndice central del intervalo
  inf: límite inferior del intervalo
  sup: límite superior del intervalo

Agoritmo:
inf = 0
sup = tam-1

Mientras (inf <= sup) hacer
  // División entera: se trunca la fracción
  centro = ((sup - inf) / 2) + inf
  Si vec[centro] == dato
     devolver centro;
  Sino
   Si dato < vec[centro]
    sup = centro – 1;
   Sino
    inf = centro + 1;
   Fin Si
  Fin Si
Fin Mientras
Devolver -1
```

Ejercicio Auto-evaluación A5.14: *Realizar el algoritmo de búsqueda binaria en Java y probarlo con el mismo array del ejercicio anterior pero ordenado:*
{-30, -27, -23, -21, -19, -19, -16, -13, -6, -6, 8, 9, 14, 20, 20, 21, 22, 24, 25, 27}
Buscar el número -21, y después el 9. Observar los resultados obtenidos.
Cuando el algoritmo acaba la ordenación debe indicarnos la cantidad de elementos visitados hasta que encuentra el que buscamos.

5.2 Arrays multidimensionales.

Los arrays multidimensionales tienen una manera casi identica de definirse a los

unidimensionales. La sistemática con la que se alojan en memoria es similar también.

Dentro de la categoría de array multidimensional entran los bidimensionales, tridimensionales, etc...

Definimos un array bidimensional de enteros por ejemplo de la siguiente manera:

```
int bidi[][]=new int [3][5];
```

La primera diferencia notable que se puede observar es que al tener dos dimensiones tiene dos corchetes dobles. La segunda es que para cada una de las dimensiones tenemos que definir su longitud. Por tanto lo que al final tenemos es un array de 3x5=15 celdas enteras. Ese array tendrá un tamaño en memoria de 3x5x4Kbytes=60Kbytes. Por cada celda de la primera dimensión tenemos 5 celdas enteras, si quisieramos representar esto de una manera más gráfica sería la siguiente:

		Columnas				
		0	1	2	3	4
Filas	0	0	0	0	0	0
	1	0	0	0	0	0
	2	0	0	0	0	0

Supongamos que ahora queremos meter valores, para realizarlo tenemos que hace lo siguiente:

```
bidi[1][3]=7;
bidi[0][4]=-3;
bidi[2][2]=9;
bidi[1][0]=1;
```

El resultado de esto será el siguiente:

		Columnas				
		0	1	2	3	4
Filas	0	0	0	0	0	-3
	1	1	0	0	7	0
	2	0	0	9	0	0

Como vemos podemos usar el concepto de “filas y columnas” para ubicarnos en el array. Para calcular la longitud de cada una de ellas podemos hacer lo siguiente:

```
int tamFilas=bidi.length;        // Así calculo el tamaño de las filas
int tamColumnas=bidi[0].length; // Así calculo el tamaño de las columnas
```

Ahora, una de las cuestiones más relevantes que veremos en este apartado es como recorrer este tipo de arrays. La respuesta a esto no es compleja, ya que consiste en usar bucles anidados, generalmente para recorrer los arrays usaremos bucles “for”. Para recorrer este array que estamos usando de ejemplo lo haremos de la siguiente manera:

```
        for(fila=0;fila<tamFilas;fila++){
            for(columna=0;columna<tamColumnas;columna++){
            // Importante hacerlo con print para que se vea bien
                System.out.print(bidi[fila][columna]+" ");
            }
            // Importante hacer aquí un println para que baje de
            // línea una vez hemos impreso una fila
            System.out.println();
        }
```

Como vemos consiste en dos bucles anidados, empieza la "fila" en cero y recorro todas las columnas y las muestro en el segundo "for", después vuelvo de nuevo al "for" inicial, y empieza con la "fila" 1, después recorre todas sus columnas, y así sucesivamente, como vemos para que cada "fila" nos la muestre en una línea diferente lo que hago es hacer un "println". Esto que acabamos de hacer podríamos hacerlo en un método "mostrar" al cual le pasamos un array bidimensional y nos lo muestra.

Por tanto el código nos queda de la siguiente manera:

```
public class bidimensional {
    public static void mostrar(int array[][]){
        int fila,columna,tamFilas,tamColumnas;
        tamFilas=array.length; // Tamaño de las filas
        tamColumnas=array[0].length; // Tamaño de las columnas
        for(fila=0;fila<tamFilas;fila++){
            for(columna=0;columna<tamColumnas;columna++){
            // Importante hacerlo con print para que se vea bien
                System.out.print(array[fila][columna]+" ");
            }
            // Importante hacer aquí un println para que baje de
            // línea una vez hemos impreso una fila
            System.out.println();
        }
    }
    public static void main(String[] args) {
        int bidi[][]=new int [3][5];
        bidi[1][3]=7;
        bidi[0][4]=-3;
        bidi[2][2]=9;
        bidi[1][0]=1;
        mostrar(bidi);
    }
}
```

Como se puede observar en el método "mostrar" se le pasa un array bidimensional y lo hemos indicado con "int array[][]". Si quisieramos que devuelva un array la cabecera del método sería el siguiente:

```
public static int[][] void mostrar(int array[][])
```

Si quisieramos dotar a un array bidimensional de valores directamente habría que hacer algo como esto:

```
int a1[][]=
          {
                    {2,8,4,1,7,9,3},
                    {4,1,2,5,7,4,1},
                    {2,8,4,1,7,9,3},
                    {4,1,2,5,7,4,1},
                    {7,2,11,54,4,1,1}
          };
```

Todo esto que hemos realizado es para un array de dos dimensiones, imaginemos que queremos hacer lo mismo con un array de tres dimensiones, entonces habría que usar 3 bucles “for” anidados.

Veamos un ejemplo:

```
public class tridimensional {
     public static void main(String[] args) {
          int a3[][][]=new int[3][2][5];
          int f=0,c=0,z=0;
          int tamf=a3.length;
          int tamc=a3[0].length;
          int tamz=a3[0][0].length;
          int tamTotal=tamf*tamc*tamz;
          for(f=0;f<tamf;f++){
```

```
                    for(c=0;c<tamc;c++){
                          for(z=0;z<tamz;z++){
                             System.out.println(tamTotal+"Fila "+f+" Col "+c+
                                    " Valor "+z+":"+a3[f][c][z]);
                          }
                    }
              }
        }
}
```

Ejercicio Auto-evaluación A5.15: *Realizar un programa en el cual tenemos el array bidimensional con los siguientes valores:*

```
int a1[][]={
                    {2,8,4,1,7,9,3},
                    {4,1,2,5,7,4,1},
                    {2,8,4,1,7,9,3},
                    {4,1,2,5,7,4,1},
                    {7,2,11,54,4,1,1}
              };
```

Realizar Crear un método llamado "sumaFilas" que nos suma las filas y nos muestra el resultado.

Ejercicio Auto-evaluación A5.16: *Realizar para el ejercicio anterior un método llamado "sumaColumnas" que nos suma las columnas y nos muestra el resultado.*

Ejercicio Auto-evaluación A5.17: *Crear un método "dibujo1" que nos rellena un array con los siguientes datos.*

```
XXXXXOOOOOO
XXXXXOOOOOO
XXXXXOOOOOO
XXXXXOOOOOO
XXXXXOOOOOO
XXXXXOOOOOO
XXXXXOOOOOO
```

Primero con "X" mayúsculas y la otra mitad con "O" mayúsculas. Crear un método "mostrarFigura" que nos muestra la figura por pantalla.

5.3 Tipos Abstractos de Datos (TAD): Pilas, Colas y Listas.

Los tipos abstractos de datos son estructuras que se usan para almacenar datos. Cada una de estas estructuras tiene un comportamiento diferente dependiendo de si estamos usando una "Pila", una "Cola" o una "Lista".

Una cosa que tendrán todos ellos en común es que tendrán los siguientes métodos:

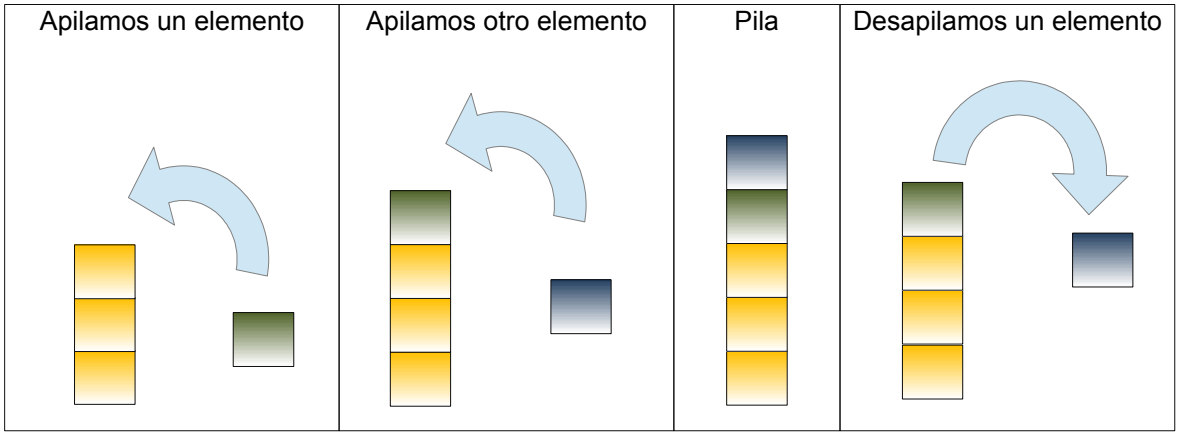
- *Insertar*: Se le pasa como parámetro el elemento que queremos insertar en la

estructura.

- *Eliminar*: Elimina el elemento que corresponda en la estructura.
- *Mostrar*: Muestra los elementos de la estructura por pantalla.

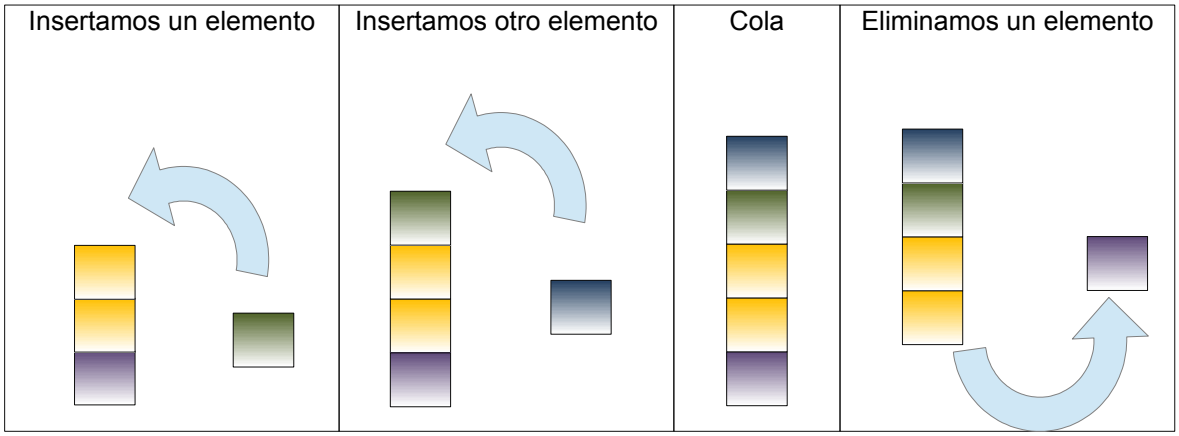
Pila

Una estructura de pila es aquella en la cual se van apilando elementos y desapilando, de tal manera que el último elemento en entrar sería el primero en salir. Vamos a ver un ejemplo gráfico:



Cola

Una estructura de cola es aquella en la cual se van agregando elementos por el final y se eliminan por el principio, de tal manera que el primer elemento que entró sería el primero en salir. Vamos a ver un ejemplo gráfico:

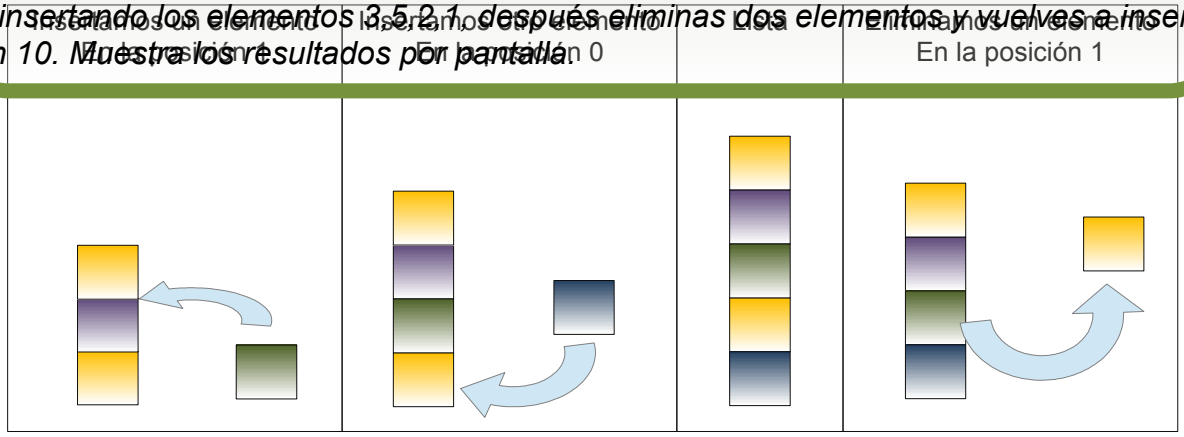


Lista

Una estructura de lista es aquella en la cual se van agregando elementos por en la posición que deseemos y se eliminan en la posición que deseemos, de tal manera que debemos pasarle las posiciones en las que queremos insertar y eliminar elementos. Vamos a ver un ejemplo gráfico:

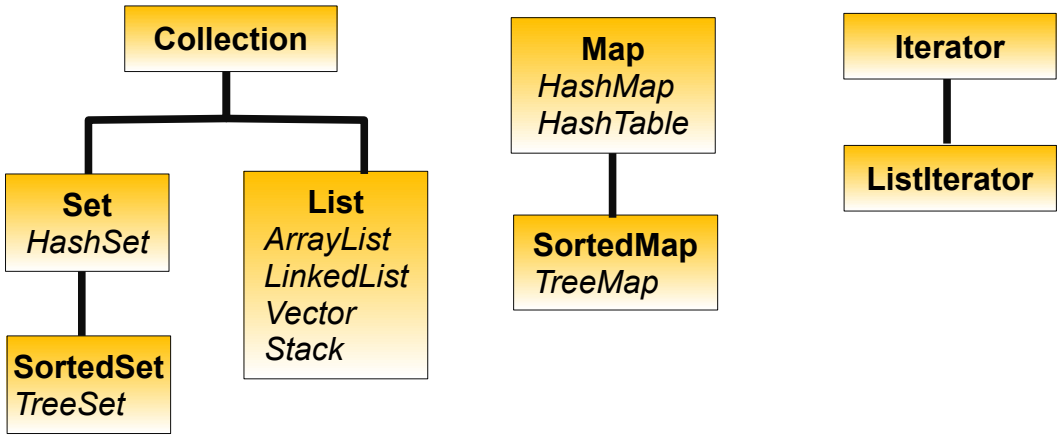
Ejercicio Auto-evaluación A5.18: *Conociendo el funcionamiento de una Pila, Implementa una clase "Pila" y después úsala creando una "Pila" de hasta 7 elementos e insertando los elementos 3,5,2,1, después eliminas dos elementos y vuelves a insertar un 10. Muestra los resultados por pantalla.*

Ejercicio Auto-evaluación A5.19: *Conociendo el funcionamiento de una Cola, Implementa una clase "Cola" y después úsala creando una "Cola" de hasta 7 elementos e insertando los elementos 3,5,2,1, después eliminas dos elementos y vuelves a insertar un 10. Muestra los resultados por pantalla.*



5.4 Colecciones

En java disponemos de una serie se herramientas u objetos que nos permiten guardar otros objetos como si de un array se tratase, con el matiz de que esta vez no son estáticos, sino que son dinámicos, esto nos aporta la ventaja de que inicialmente no tenemos que saber el tamaño que va a tener, puede crecer o disminuir en tiempo de ejecución, por ello es dinámico. Cada una de estas colecciones aunque tienen todas ellas una naturaleza homogenea hay ciertos aspectos que las diferencian. Todas provienen de una clase llamada "Collection" y de ella derivan otras, de tal manera que podemos usar la implementación de una cola, una pila, una lista, conjuntos y mapas. Cada uno cumple unas características diferentes. Veamos un esquema de como están distribuidos:



Vamos a ver según cada una de las interfaces que usaremos (Set, List y Map), cuales son las implementaciones que debemos usar según los resultados que deseemos:

		Implementaciones				
		Tabla Hash	*Array redimensionable*	*Arbol balanceado*	*Listas enlazadas*	*Tabla Hash + Listas enlazadas*
Interfaces	*Set*	HashSet		TreeSet		LinkedHashSet
	List		ArrayList Vector Stack		LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap

Los “Hash” como veremos, son funciones y objetos, que reciben unos elementos y los transforman en una salida computada mediante un algoritmo.

Todas las interfaces vistas arriba disponen a su vez de una interfaz común para recorrerlas, esta es la interfaz “Iterator” y “ListIterator”.

Los métodos principales proporcionados por la interfaz “Collection” son los siguientes:

Método	*Funcionalidad*
int size()	Devuelve el número de objetos de la colección
boolean add(E elemento)	Agrega el elemento de clase E a la colección. Devuelve *“true”* si la colección ha cambiado tras la operación.
boolean addAll(Collection c)	Agrega una colección c a la colección existente. Devuelve *“true”* si la colección ha cambiado tras la operación.
boolean remove(Object elemento)	Elimina el elemento que coincida en la colección. Devuelve *“true”* si la colección ha cambiado tras la operación.
Boolean removeAll(Collection c)	Elimina la colección c que coincida con las colección existente. Devuelve *“true”* si la colección ha cambiado tras la operación.
Iterator <E> iterator()	Devuelve un iterador para recorrer los elementos de la colección.
void clear()	Elimina todos los elementos de la colección

Método	Funcionalidad
boolean contains(Object elemento)	Devuelve “true” si encuentra el elemento en la colección
boolean contains(Collection c)	Devuelve “true” si encuentra la colección c en la colección actual.
Object[ ] toArray()	Devuelve un array estático con todos los elementos de la colección.
boolean isEmpty()	Devuelve “true” si la colección está vacía, en caso contrario devuelve “false”.

5.4.1 Listas (List)

En este apartado veremos ArrayList y LinkedList.

ArrayList se usa de forma similar a un array y LinkedList funciona como una lista enlazada. La principal diferencia entre ambos es la siguiente:

- En ArrayList aunque funciona como un array no hay que preocuparse por el tamaño y es más eficiente que LinkedList para manejar los datos.
- LinkedList permite manejar los objetos como si fuese una lista, una cola o una pila.

Veamos un ejemplo de como funciona un “ArrayList”, imaginemos que queremos guardar una serie de palabras en un array pero no sabemos o no queremos que tenga un tamaño determinado. Además en el ejemplo se ilustrarán las 3 maneras en las que podemos recorrer el array:

```
import java.util.ArrayList;
import java.util.Iterator;

public class Listas1 {
	public static void main(String[] args) {
		ArrayList<String> al=new ArrayList<String>();
		al.add("Hola");
		al.add("esto");
		al.add("es");
		al.add("un");
		al.add("ejemplo");
		System.out.println("El array tiene "+al.size()+" palabras.");
		System.out.println("Voy a mostrar las 3 maneras de
recorrerlo:");
		System.out.println("1. Mediante un 'for' normal:");
		for(int i=0;i<al.size();i++){
			System.out.println("Palabra "+i+": "+al.get(i));
		}
		System.out.println("2. Mediante un 'for' que recorre
objetos:");
		for(String palabra : al){
			System.out.println("Palabra: "+palabra);
		}
		System.out.println("3. Mediante la interfaz 'Iterator':");
```

```
            Iterator i=al.iterator();
            while(i.hasNext()){
                  System.out.println("Palabra: "+i.next());
            }
      }
}
```

Para recorrer las colecciones siempre el método más correcto será usar un “Iterator” o un “ListIterator”. Los métodos más relevantes de “Iterator” y “ListIterator” son:

Iterator	
Método	*Funcionalidad*
boolean hasNext()	Devuelve “true” en caso de que exista un elemento siguiente en la colección.
Object next()	Devuelve el elemento siguiente como un objeto Object, por tanto sería necesario hacer un casting a su tipo correspondiente.
void remove()	Elimina el objeto último objeto que hemos visitado con “next()”

ListIterator	
Método	*Funcionalidad*
boolean hasNext() *Object next()* *void remove()*	Los mismos que “Iterator”.
boolean hasPrevious()	Devuelve “true” en caso de que exista un elemento anterior en la colección.
Object previous()	Devuelve el elemento anterior como un objeto Object, por tanto sería necesario hacer un casting a su tipo correspondiente.
void add(Object E)	Agrega el elemento “Object” en el último objeto que hemos visitado con “next()” o “pevious()”.
void set(Object E)	Sustituye el último objeto que hemos visitado con “next()” o “pevious()” por el elemento “Object” pasado como parámetro.
int nextIndex() *int previousIndex()*	Nos devuelve los indices siguiente y anterior respectivamente.

A continuación se va a exponer los métodos más relevantes de “ArrayList” y “LinkedList”:

ArrayList	
Método	*Funcionalidad*
void add(int pos,E elemento)	Agrega el elemento de clase E en la posición indicada.
E remove(int pos)	Elimina el elemento en la posición indicada. Devuelve el elemento de clase E eliminado o null sino ha eliminado ninguno.
E get(int pos)	Devuelve el elemento de clase E de la posición indicada.
E set(int pos, E elemento)	Sustituye el elemento de clase E en la posición indicada y lo devuelve.

ArrayList	
Método	*Funcionalidad*
int indexOf(Object elemento)	Devuelve la posición del elemento pasado como parámetro si lo encuentra, y sino lo encuentra devuelve -1.
int lastIndexOf(Object elemento)	Igual que el anterior sólo que devuelve la última ocurrencia.

LinkedList	
Método	*Funcionalidad*
void addFirst(E elemento) *void addLast(E elemento)*	Añade el elemento al principio o al final de la lista respectivamente.
E getFirst() *E getLast()*	Devuelve el primer o último elemento de la lista respectivamente.
void removeFirst() *void removeLast()*	Elimina el primer o último elemento de la lista respectivamente.
Iterator <E> iterator()	Devuelve un iterador para recorrer los elementos de la colección.
void clear()	Elimina todos los elementos de la colección
boolean contains(Object elemento)	Devuelve “true” si encuentra el elemento en la colección
void add(int pos,E elemento) *E remove(int pos)* *E get(int pos)* *E set(int pos, E elemento)* *int indexOf(Object elemento)* *int lastIndexOf(Object elemento)*	Los mismos que para ArrayList

Para comprobar alguno de los métodos vistos veamos de nuevo otro ejemplo de “ArrayList” en el cual usamos más métodos. Supongamos que creamos una lista con varios números y vamos mostrando, y después realizamos algunas modificaciones y volvemos a mostrar. Para hacer el mostrar vamos a simplificarlo realizando un método que usaremos en varias ocasiones al cual le pasamos un “Iterator” y va recorriendo y mostrando los elementos por pantalla.

```
import java.util.ArrayList;
import java.util.Iterator;

public class Listas2 {
	public static void mostrar(Iterator it){
		int pos=0;
		while(it.hasNext()){
			System.out.println("Posición "+pos+": "+it.next());
			pos++;
		}
		System.out.println("**********************");
	}
	public static void main(String[] args) {
		ArrayList<Integer> miArray=new ArrayList<Integer>();
```

```
            miArray.add(7);
            miArray.add(2);
            miArray.add(3);
            miArray.add(1);
            miArray.add(10);
            mostrar(miArray.iterator());
            // Elimino el elemento en la posición 2
            miArray.remove(2);
            mostrar(miArray.iterator());
            // Sustituyo el elemento en la pos 0, por un 20
            miArray.set(0, new Integer(20));
            mostrar(miArray.iterator());
            // Agrego el Integer 50 en la posición 1
            miArray.add(1,new Integer(50));
            mostrar(miArray.iterator());
      }
}
```

Veamos ahora también como puedo agregar una colección a otra, y vemos al tiempo el uso de “LinkedList”.

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Iterator;

public class Listas3 {
      public static void mostrar(Iterator it, String cabecera){
            int pos=0;
            System.out.println(cabecera);
            while(it.hasNext()){
                  System.out.println("Posición "+pos+": "+it.next());
                  pos++;
            }
            System.out.println("**********************");
      }
      public static void main(String[] args) {
            ArrayList<Integer> miArray=new ArrayList<Integer>();
            LinkedList<Integer> miLista=new LinkedList<Integer>();
            miArray.add(7);
            miArray.add(2);
            miArray.add(3);
            miArray.add(1);
            miArray.add(10);
            miLista.add(-5);
            miLista.add(-2);
            miLista.addFirst(-7);
            miLista.addLast(-10);
            mostrar(miArray.iterator(),"miArray\n========");
            mostrar(miLista.iterator(),"miLista\n========");
```

```
            // Agrego la lista al array
            miArray.addAll(miLista);
            mostrar(miArray.iterator(),"miArray+miLista\n==============");
            // Cambio el sexto elemento por un 100
            miArray.set(6, new Integer(100));
            // Elimino los elementos de miLista en miArray
            miArray.removeAll(miLista);
            mostrar(miArray.iterator(),"miArray-miLista\n==============");
      }
}
```

Ordenando Arrays Dinámicos

Aunque en el siguiente apartado de conjuntos veremos que es posible usar una estructura de arbol balanceado con un "TreeSet" de manera que conforme insertamos elementos estos se van ordenando, en los arrays que acabamos de ver también existe la posibilidad de ordenarlos. Esto podemos realizarlo gracias al método estático "sort" que nos proporciona la clase "Collections". El método "sort" recibe una colección y la ordena. También existe el método "reverse" que realiza la ordenación de manera inversa. Además podemos mostrar por pantalla los arrays pasándolos directamente al "System.out.println()". Veamos todo esto con un ejemplo un ejemplo:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

public class Listas4 {
      public static void ordenar1(ArrayList<Integer> o1)
      {
            Collections.sort(o1);
            Iterator i=o1.iterator();
            while(i.hasNext())
            {
                  System.out.print(i.next()+", ");
            }
            System.out.println("\n**********************");
      }

      public static void main(String[] args) {
            ArrayList<Integer> al1= new ArrayList<Integer>();
            al1.add(0);
            al1.add(3);
            al1.add(2);
            al1.add(10);
            al1.add(-4);
            al1.add(3);
            al1.add(3);
            al1.add(-10);
            al1.add(-4);
            al1.add(7);
```

```
            al1.add(1);

            System.out.println(al1);
            System.out.println("**********************");
            ordenar1(al1);
            Collections.reverse(al1);
            System.out.println(al1);
        }
}
```

Ejercicio Auto-evaluación A5.20: *Conociendo los métodos de "LinkedList", implementa una clase "PilaDinamica" y una clase "ColaDinamica" de números enteros. Para ponerla a prueba realiza los mismos pasos que en los ejercicios A5.18 y A5.19.*

5.4.2 Conjuntos (Set)

En este apartado veremos el funcionamiento de los conjuntos, concretamente las clases "*HashSet*" y "*TreeSet*". Al tratarse de conjuntos la característica principal es que no guardan elementos repetidos. Si se intenta añadir un elemento que está repetido lo devuelve con un "false".

Las diferencias principales entre "*HashSet*" y "*TreeSet*" son las siguientes:

- ***"HashSet"*** realiza las operaciones habituales de manera independiente del número de elementos. Si tiene muchos elementos es muy costoso recorrerlo con un "Iterator". Tampoco garantiza el orden, de hecho el orden puede ir variando según vamos insertando elementos.
- ***"TreeSet"*** realiza las operaciones habituales pero dependiendo del número de elementos. Además mantiene todos los elementos ordenados en un orden natural o como se le especifique en el *"Comparator"*.

Veamos los métodos mas relevantes de cada uno de ellos:

HashSet	
Método	*Funcionalidad*
Los mismos definidos en Collection	Los mismos definidos en Collection.

TreeSet	
Método	*Funcionalidad*
E first() *E last()*	Devuelve el menor y mayor, respectivamente, de los elementos del conjunto ordenado.
SortedSet<E> headSet(E elemento) *SortedSet<E> tailSet(E elemento)*	Devuelve el conjunto de elementos que son menores que, o mayores o iguales que, respectivamente, que el elemento del parámetro.

A continuación veamos un ejemplo de "HashSet" y "TreeSet", parecido al primero que

vimos para los “ArrayList” al cual agregabamos palabras, sólo que esta vez vamos a repetir palabras y veremos que no se agregan y además veremos como el “TreeSet” las ordena alfabeticamente.

```
import java.util.HashSet;
import java.util.Iterator;
import java.util.TreeSet;

public class base {
	public static void mostrar(Iterator it, String cabecera){
		int pos=0;
		System.out.println(cabecera);
		while(it.hasNext()){
			System.out.println("Posición "+pos+": "+it.next());
			pos++;
		}
		System.out.println("**********************");
	}
	public static void main(String[] args) {
		HashSet<String> hs=new HashSet<String>();
		TreeSet<String> ts=new TreeSet<String>();
		// Agrego palabras al HashSet
		hs.add("Hola");
		hs.add("esto");
		// Agrego de nuevo el elemento "esto", pero al estar
		// repetido no lo agrega, los conjuntos no tienen repetidos
		hs.add("esto");
		hs.add("es");
		hs.add("un");
		hs.add("ejemplo");
		// Como se podrá observar el orden cambia, no hay orden
		mostrar(hs.iterator(),"HashSet de "+hs.size()+" palabras." +
				"\n====================================");
		// Ahora agrego los mismos elementos al TreeSet
		ts.addAll(hs);
		// Como se podrá observar están ordenados alfabeticamente
		mostrar(ts.iterator(),"TreeSet de "+ts.size()+" palabras." +
				"\n====================================");
	}
}
```

5.4.3 Mapas (Map)

Los Mapas o “Map” es una estructura de datos agrupada en parejas Clave/Valor. Podriamos considerarlos como una tabla con dos columnas, en la cual a cada clave le corresponde su valor en la siguiente columna. La clave debe ser única ya que se utiliza para acceder al valor, por tanto no puede haber redundancia.

Aunque “Map” no deriva de “Collection” es posible verlos o asemejarlos también como

una colección de pares Clave/Valor. Una ventaja que nos aporta "Map" es que las búsquedas de elementos son casi instantáneas debido a que accedo al elemento a través del mapeado de la clave. Otra característica importante es que un **"Map" no se puede recorrer con un "Iterator"**, debido a que no es una secuencia de elementos, sino una tabla de mapeo Clave/Valor. **Tampoco puede ordenarse con el método "sort" de "Collections"**. Para mantener un "Map" ordenado debemos usar la implementación "TreeMap" de "SortedMap".

Las implementaciones que veremos serán las de "HashMap" y "TreeMap". Los métodos más relevantes que nos proporciona la interfaz "Map" son los siguientes:

Map	
Método	*Funcionalidad*
Object put(Object clave,Object valor)	Inserta el par "clave/valor" en el "Map".
void putAll(Map M)	Vuelca los pares "clave/valor" de M en el "Map" sobre el que ejecutamos el método.
Set keySet()	Devuelve un conjunto con las claves
Collection values()	Devuelve una colección con los valores
Object get(Object clave)	Devuelve el objeto correspondiente a esa clave
Object remove(Object clave)	Elimina el objecto correspondiente a esa clave y lo devuelve.

Veamos un ejemplo de uso de "HashMap" y "TreeMap", como se podrá observar en el ejemplo es aconsejable cuando creamos el "Map" indicarle el tipo de la "clave", y el tipo del "valor" que en este caso son iguales, y lo indicamos poniendo <String,String>:

```
import java.util.HashMap;
import java.util.TreeMap;

public class base {
      public static void main(String[] args) {
            HashMap<String,String> hm = new HashMap<String,String>();
            hm.put("Programación", "Andrés");
            hm.put("Entornos", "Víctor");
            hm.put("Inglés", "Gabriel");
            hm.put("FOL", "Oscar");
            System.out.println("HASHMAP de "+hm.size()+
                        " elementos\n*******************************");
            System.out.println("CLAVES:"+hm.keySet());
            System.out.println("VALORES:"+hm.values());
            System.out.println("CLAVE=VALOR:"+hm);
            System.out.println("¿Quién imparte programación?->"+
                        hm.get("Programación"));

            TreeMap<String,String> tm = new TreeMap<String,String>();
            tm.put("Programación", "Andrés");
            tm.put("Entornos", "Víctor");
            tm.put("Inglés", "Gabriel");
```

```
            tm.put("FOL", "Oscar");
            System.out.println("\nTREEMAP de "+tm.size()+
                        " elementos\n*****************************");
            System.out.println("CLAVES:"+tm.keySet());
            System.out.println("VALORES:"+tm.values());
            System.out.println("CLAVE=VALOR:"+tm);
            System.out.println("¿Quién imparte inglés?->"+
                        tm.get("Inglés"));
      }
}
```

5.5 Ejercicios Propuestos

1. Realizar un programa en el cual tenemos un array con los números: 3,2,-1,5,7,10 y los muestra por pantalla.

2. Realizar un programa que recibe un número por teclado y nos guarda los 5 primeros múltiplos de ese número y los muestra.
Ejemplo:
Dame un número: 3
3, 6, 9, 12, 15,

3. Realizar un programa que nos pide 5 palabras y nos las muestra en el orden inverso al que las hemos introducido.

4. Realizar un programa que nos pide 5 números y nos muestra los siguientes resultados en el siguiente orden:
La suma de todos los números.
La resta de todos los números.
La multiplicación de todos los números.
La división de todos los números.

5. Realizar un programa que nos pide 5 números y nos guarda cada uno de esos números multiplicados por 3 en otro array de 5 números.

6. Realizar un programa que recibe 5 números positivos y 5 negativos de manera aleatoria y guarda en un array los positivos y en otro los negativos.

7. Realizar un método que se le pasa un array unidimensional (del tamaño que el alumno quiera) y lo rellena con valores aleatorios.

8. Realizar un programa que busca un número en un array aleatorio de 50 números. Si encuentra el número nos tiene que indicar en que posición lo ha encontrado.

9. Realizar un método parecido al ejercicio 7, pero que esta vez nos rellena un array bidimensional (de 2 dimensiones, con los tamaños que el alumno quiera), con valores aleatorios.

10. Usando el ejercicio anterior realizar un programa el cual nos genera un array de dos dimensiones, de 10 filas y 20 columnas, con valores aleatorios, y a continuación pide al usuario un número de fila y otro de columna y nos muestra su valor.

11. Usando el ejercicio 9 realizar un programa el cual nos genera un array de dos dimensiones, de 10 filas y 20 columnas, con valores aleatorios, y a continuación pide al usuario un número. Para finalizar nos indica cuantos números del array son mayores y cuantos son menores que el número dado.

12. Leer por teclado una serie de 10 números enteros. La aplicación debe indicarnos si los números están ordenados de forma creciente, decreciente, o si están desordenados.

13. Crear un programa que lea por teclado una tabla de 10 números enteros y la desplace una posición hacia abajo (el último pasa a ser el primero).

14. Realizar una agenda que tenga un tamaño de 10 personas. Hay que tener un método para introducir personas, otro para buscar y otro para eliminar. Introducir personas y ponerlo a prueba.

15. Realizar un programa que guarda en un array bidimensional de 7x7 la siguiente figura:

```
XOOOOOX
OXOOOXO
OOXOXOO
OOOXOOO
OOXOXOO
OXOOOXO
XOOOOOX
```

16. Realizar un programa que crea 2 listas de números con los siguientes datos:

Lista 1: 0, 3, 2, 10, -4, 3

Lista 2: 3,-10,-4, 7, 1

Se pide usar los objetos que sean más adecuados y eficientes según lo visto en clase para los siguientes apartados (apunta el resultado observado en cada apartado). Debemos crear los siguientes métodos y mostrar el resultado:

a) Método “mezclar” que combina las dos listas en una sin que se repita ningún elemento.

b) Método “ordenar1” que combina las dos listas en una ordenada.

c) Método “ordenar2” que combina las dos listas en una ordenada decrecientemente.

d) Método “comun” que genera una lista con aquellos elementos comunes.

e) Método “diferencia” que genera una lista con aquellos elementos que no son comunes.

f) Realizar el método “mezclar” del punto “a)” de manera que lo que devuelva sea un ArrayList.

17. Realiza un programa que almacene en memoria los siguientes datos de empleados:
    - Nombre
    - Puesto de trabajo
    - DNI

Se necesita que el acceso a los datos sea instantáneo. Determina la forma y uso de clases que debes hacer para estos requisitos. Realiza un programa de ejemplo en el que introduces empleados.

18. Realizar un juego de las 3 en raya. Para ello usar un array estático de 3x3. Debe haber un turno que es generado aleatoriamente al principio. Cuando juega el jugador pone las fichas de tipo “O”, y cuando juega la máquina pone fichas de tipo “X”. El juego acaba cuando uno de los dos jugadores gana o hay un empate.

5.6 Solución a los ejercicios de Auto-evaluación

Solución Auto-evaluación A5.1:

```
public class Autoeval51 {
	public static void ejer51(int arr[]){
		int i;
		for(i=0;i<100;i++){
			arr[i]=i;
		}
	}

	public static void mostrar(int arr[],int tam){
		int i=0;
		for(i=0;i<tam;i++){
			System.out.println("Pos "+i+" elemento:"+arr[i]);
		}
	}
}
```

Y el main:

```
	public static void main(String[] args) {
		int arr[]=new int[100];
		ejer51(arr);
		mostrar(arr,100);
	}
```

Solución Auto-evaluación A5.2:

```
public class Autoeval52 {
	public static void ejer52(int arr[]){
		int i;
		for(i=0;i<100;i++){
			arr[i]=i;
		}
	}

	public static void mostrar_inverso(int arr[],int tam){
		int i=0;
		for(i=tam-1;i>=0;i--){
			System.out.println("Pos "+i+" elemento:"+arr[i]);
		}
	}
}
```

Y el main:

```
	public static void main(String[] args) {
		int arr[]=new int[100];
		ejer52(arr);
		mostrar_inverso(arr,100);
	}
```

Solución Auto-evaluación A5.3:

La solución sería simplemente cuando llamamos en el main al método "incrementaNotas" hacerlo de la siguiente manera: `incrementaNotas(notas.clone());`

Solución Auto-evaluación A5.4:

```
public class Autoeval54 {
	public static void aleatorio(int arr[],int tam){
		int i=0;
		for(i=0;i<tam;i++){
			arr[i]=(int)(Math.random()*tam);
		}
	}
	public static void ejer54(int arr[],int tam){
		System.out.println("El primer elemento es:"+arr[0]);
		System.out.println("El ultimo elemento es:"+arr[tam-1]);
	}
}
```

Y el main:

```
	public static void main(String[] args) {
		int arr[]=new int[100];
		aleatorio(arr,100);
		ejer54(arr,100);
	}
```

Solución Auto-evaluación A5.5:

```
public class Autoeval55 {
	public static void aleatorio(int arr[],int tam){
		int i=0;
		for(i=0;i<tam;i++){
			arr[i]=(int)(Math.random()*tam);
		}
	}
	public static void ejer55(int arr[],int tam,int inf,int sup){
		int i=0;
		for(i=0;i<tam;i++){
			if(arr[i]>=inf && arr[i]<=sup){
				System.out.print(arr[i]+",");
			}
		}
	}
}
```

Y el main:

```
	public static void main(String[] args) {
		int arr[]=new int[100];
		aleatorio(arr,100);
		ejer55(arr,100,4,20);
	}
```

Solución Auto-evaluación A5.6:

```
public class Autoeval56 {
	public static void mostrar(int arr[],int tam){
		int i=0;
		for(i=0;i<tam;i++){
			System.out.println("Pos "+i+" elemento:"+arr[i]);
		}
	}
	public static void ejer56(int a1[],int a2[],int a3[]){
		int i=0;
		for(i=0;i<7;i++){
			if(a1[i]>a2[i]){
				a3[i]=a1[i];
			}else{
				a3[i]=a2[i];
			}
		}
	}
}
```

Y el main:

```
	public static void main(String[] args) {
		int a1[]={1,4,3,7,10,9,40};
		int a2[]={20,2,30,34,0,-10,4};
		int a3[]=new int[7];

		ejer56(a1,a2,a3);
		mostrar(a3,7);
	}
```

Solución Auto-evaluación A5.7:

```
public class Autoeval57 {
	public static void mostrar(char arr[],int tam){
		int i=0;
		for(i=0;i<tam;i++){
			System.out.println("Pos "+i+" elemento:"+arr[i]);
		}
	}
	public static char[] ejer57a(String p1,String p2){
		char arr[]=new char[p1.length()+p2.length()+1];
		int i=0,j=0;
		for(i=0;i<p1.length();i++){
			arr[i]=p1.charAt(i);
		}
		arr[i++]=' ';
		for(j=0;j<p2.length();j++){
			arr[i+j]=p2.charAt(j);
		}
		return arr;
```

```
	}

	public static char[] ejer57b(String p1,String p2){
		String total=p1+" "+p2;
		int i=0;
		char arr[]=new char[total.length()];
		for(i=0;i<total.length();i++){
			arr[i]=total.charAt(i);
		}
		return arr;
	}
}
```

Y el main:

```
	public static void main(String[] args) {
		char ar[]=new char[10];

		ar=ejer57a("hola","adios");
		mostrar(ar,10);
		ar=ejer57b("adios","hola");
		mostrar(ar,10);
	}
```

Solución Auto-evaluación A5.8:

```
public class Autoeval58 {
	public static void mostrar(String arr[],int tam){
		int i=0;
		for(i=0;i<tam;i++){
			System.out.println("Pos "+i+" elemento:"+arr[i]);
		}
	}
	public static void ejer58a(char e7[]){
		for(int i=0;i<e7.length;i++){
			if(e7[i]!=' '){
				System.out.print(e7[i]);
			}else{
				System.out.println();
			}
		}
		System.out.println();
	}
	public static void ejer58b(char e7[]){
		String pal="";
		for(int i=0;i<e7.length;i++){
			if(e7[i]!=' '){
				pal=pal+e7[i];
			}else{
				System.out.println(pal);
				pal="";
```

```
                }
            }
            System.out.println(pal);
        }

        public static String[] ejer58c(char e7[]){
            String palabras[];
            String pal="";
            int numPal=1,palabra=0;
            for(int i=0;i<e7.length;i++){
                if(e7[i]==' '){
                    numPal++;
                }
            }
            palabras=new String[numPal];
            for(int i=0;i<e7.length;i++){
                if(e7[i]!=' '){
                    pal=pal+e7[i];
                }else{
                    palabras[palabra++]=pal;
                    pal="";
                }
            }
            palabras[palabra++]=pal;
            return palabras;
        }
}
```

Como se podrá obervar se han dado hasta 3 soluciones, algunas de ellas más complejas, en la solución "ejer58c" por ejemplo tenemos una solución compleja en la que guardamos todas las palabras separadas dentro de un array de String.

El main quedaría así:

```
    public static void main(String[] args) {
        char e8[]={'e','s','t','o',' ','e','s',' ','u','n','a',' ',
                'p','r','u','e','b','a'};
        ejer58a(e8);
        ejer58b(e8);
        String e58c[]=ejer58c(e8);
        mostrar(e58c,e58c.length);
    }
```

Solución Auto-evaluación A5.9:

```
public class Autoeval59 {

    public static void mostrar(int arr[]){
        int tam=arr.length;
        int i=0;
        for(i=0;i<tam;i++){
            System.out.println("Pos "+i+" elemento:"+arr[i]);
```

```
		}
	}

	public static int[] ejer59(int e8[]){
		int tam=(int)((float)e8.length/2);
		int a8[]=new int[tam];
		for(int i=0,j=e8.length-1;i<j;i++,j--){
			a8[i]=e8[i]+e8[j];
		}
		return a8;
	}

	public static void main(String[] args) {
		int e59[]={10,2,3,-5,8,1,50,-3};
		int a59[]=ejer59(e59);

		mostrar(e59);
		mostrar(a59);
	}
}
```

Solución Auto-evaluación A5.10:

Para resolverlo a parte de lo que nos pide el ejercicio (el número de iteraciones), también vamos a calcular el tiempo que tarda en segundos en resolverlo. Para calcular el tiempo vamos a usar el la llamada "`System.currentTimeMillis()`" que nos devuelve el tiempo actual en milisegundos, lo que hacemos es llamar a este método antes de hacer la ordenación y guardarlo en una variable llamada "tiempoInicial" de tipo "long", y cuando acabamos de ordenar restamos el tiempo actual "`System.currentTimeMillis()`" menos el "tiempoInicial" y esto nos devolverá el tiempo total que hemos tardado. Como para un array como el que nos piden, de tan solo 5 elementos no tiene sentido medir el tiempo, ya que es casi instantaneo también se va a crear un array de 50.000 números enteros, que va a ser iniciado con valores aleatorios, para ello se va a crear también una función "arrayAleatorio" para que rellene el array con esos valores aleatorios y se ordenará también este array para observar el tiempo e iteraciones transcurridas. **Atención!:** el array de 50.000 enteros es posible que no sea soportado por su máquina, ya que hay que tener en cuenta que el tamaño de cada entero son 4 Kbytes, si tenemos en cuenta que son 50.000 enteros, estaríamos hablando de alojar en memoria un tamaño total de 50.000x4 = 200.000 Kbytes, lo que aproximadamente equivale a 200 Mbytes de memoria, además de la carga que esto conlleva para el procesador al realizar la ordenación. Si es su caso es suficiente con bajar la cantidad de enteros, pruebe con 10.000 enteros o con 5.000.

Además al método de ordenación le pasaremos un valor de tipo "boolean" llamado "mostrar" para indicar si queremos ver la resolución iteración a iteración o simplemente queremos ver el resultado final que en tal caso lo pondremos a "false". Esto nos servirá porque en el caso de un array pequeño se puede observar el proceso, pero en un array muy grande puede ser tedioso, y es preferible mostrar solo el resultado final.

Una vez dicho esto procedemos con el código:

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

```
public class Sort {
	// Esta variable la usamos para pulsar intro y continuar
	static Scanner sc=new Scanner(System.in);

	// Generamos un array con números aleatorios entre el valor
	// inferior y superior
	public static void arrayAleatorio(int a[],int inf,int sup){
		int pos;
		int tam=a.length;
		for (pos=0;pos<tam;pos++){
			a[pos]=(int)(Math.random()*(sup-inf))+inf;
		}
	}
	// Método para intercambiar los valores en el array
	public static void intercambio(int array[],int pos1,int pos2){
		int temp=array[pos1];
		array[pos1]=array[pos2];
		array[pos2]=temp;
	}
	// Método para mostrar los valores por pantalla
	public static void mostrar(int a[]){
		for(int i=0;i<a.length;i++){
			System.out.print(a[i]+", ");
		}
		System.out.println();
	}
	// Algoritmo de ordenación SORT
	public static void seleccionSort(int a[], boolean mostrar){
		int posMenor,i,j;
		long tiempoInicial;

		System.out.print("ORDENACIÓN SORT: ");
		mostrar(a);
		// Iniciamos el tiempo
		tiempoInicial=System.currentTimeMillis();
		// Acaba cuando está ordenado
		for(i=0;i<a.length-1;i++){
			// Suponemos que el menor es el primero que encontramos
			posMenor=i;
			// Recorremos el array buscando si hay alguno menor
			for(j=i+1;j<a.length;j++){
				if(a[j]<a[posMenor]){
					posMenor=j;
				}
			}
			// Intercambio
			intercambio(a,i,posMenor);
```

```
                // Mostramos los valores
                if (mostrar==true){
                      System.out.print("ITERACION "+(i+1)+":");
                      mostrar(a);
                }
          }
          System.out.print("Resuelto en:"+i+" iteraciones y "+
              ((System.currentTimeMillis()-tiempoInicial)/(float)1000)+
               " segundos -> ");
          mostrar(a);
     }

     public static void main(String[] args) {
          int arrayEjemplo[]={7,4,5,-1,1};
          int arrayExtremo[]=new int[50000];
          arrayAleatorio(arrayExtremo,-30000,30000);
          seleccionSort(arrayEjemplo,true);
          System.out.print("Pulsa intro para seguir...");
          sc.nextLine();
          seleccionSort(arrayExtremo,false);
     }
}
```

Solución Auto-evaluación A5.11:

Para solucionar este ejercicio seguiremos la misma dinámica que el anterior, tendremos un array pequeño y otro grande y observaremos tanto las iteraciones como el tiempo. De este algoritmo cabe recalcar que las iteraciones se ejecutarán mientras que "i", que es donde se van colocando los menores, y "k", que es donde se van colocando los mayores, no se crucen, o sea, mientras que "i<k", "i" empieza por el principio a colocar los menores, y "k" empieza por el final a colocar los mayores. Otro aspecto importante del agoritmo es que puede darse la casualidad que cuando intercambiamos primero el menor puede ser que ese intercambio afecte a la posición que habiamos encontrado del mayor, entonces debemos arreglarlo indicando después del cambio del menor dónde ha quedado la posición del mayor elemento.

Procedemos pues con el código:

```
import java.util.Scanner;

public class SortMejorado {
     // Esta variable la usamos para pulsar intro y continuar
     static Scanner sc=new Scanner(System.in);

     // Generamos un array con números aleatorios entre el valor
     // inferior y superior
     public static void arrayAleatorio(int a[],int inf,int sup){
          int pos;
          int tam=a.length;
          for (pos=0;pos<tam;pos++){
               a[pos]=(int)(Math.random()*(sup-inf))+inf;
```

```
		}
	}
	// Método para intercambiar los valores en el array
	public static void intercambio(int array[],int pos1,int pos2){
		int temp=array[pos1];
		array[pos1]=array[pos2];
		array[pos2]=temp;
	}
	// Método para mostrar los valores por pantalla
	public static void mostrar(int a[]){
		for(int i=0;i<a.length;i++){
			System.out.print(a[i]+", ");
		}
		System.out.println();
	}
	// Algoritmo de ordenación SORT MEJORADO
	public static void seleccionSortMejorado(int a[], boolean mostrar){
		int posMenor=0,posMayor,i,k,j;
		long tiempoInicial;

		System.out.print("SORT MEJORADO: ");
		mostrar(a);
		// Iniciamos el tiempo
		tiempoInicial=System.currentTimeMillis();
		// Acaba cuando está ordenado
		for(i=0,k=a.length-1;i<k;i++,k--){
			// Suponemos que el menor es el primero que encontramos
			// y el mayor el último
			posMenor=i;
			posMayor=k;
			// Recorro el array buscando si hay alguno menor y mayor
			for(j=i;j<a.length-i;j++){
				if(a[j]<a[posMenor]){
					posMenor=j;
				}
				if(a[j]>a[posMayor]){
					posMayor=j;
				}
			}
			// Intercambio del menor
			intercambio(a,i,posMenor);
			// En el caso que el intercambio del menor afecte a la
			// posición del mayor encontrado lo arreglamos
			if (posMayor==i){
				posMayor=posMenor;
			}else{
				if (posMayor==posMenor){
					posMayor=i;
```

```
                    }
                }
                // Intercambio del mayor
                intercambio(a,k,posMayor);
                // Mostramos los valores
                if(mostrar==true){
                    System.out.print("ITERACION "+(i+1)+":");
                    mostrar(a);
                }
            }
            System.out.print("Resuelto en:"+i+" iteraciones y "+
                ((System.currentTimeMillis()-tiempoInicial)/(float)1000)+
                 " segundos -> ");
            mostrar(a);
        }

        public static void main(String[] args) {
            int arrayEjemplo[]={7,4,5,-1,1};
            int arrayExtremo[]=new int[50000];
            arrayAleatorio(arrayExtremo,-30000,30000);
            seleccionSortMejorado(arrayEjemplo,true);
            System.out.print("Pulsa intro para seguir...");
            sc.nextLine();
            seleccionSortMejorado(arrayExtremo,false);
        }
}
```

Solución Auto-evaluación A5.12:

Para solucionar este ejercicio seguiremos la misma dinámica que el anterior, tendremos un array pequeño y otro grande y observaremos tanto las iteraciones como el tiempo. De este algoritmo cabe recalcar que las iteraciones se ejecutarán mientras que la variable de tipo “boolean” llamada “ordenado” sea igual a “false” y que no hayamos recorrido el array tantas veces como celdas tiene. La variable “ordenado” juega un papel importante, ya que cada vez que iniciamos una nueva iteración empieza siendo “true” y si realizamos algún intercambio se pone a “false”, con lo cual solo cuando no hayamos realizado ningún intercambio acabará el algoritmo de ordenación en un número de iteraciones menor que “Sort”. Lo que si se puede observar es que el tiempo de ordenación es mayor, esto es debido al coste del intercambio, ya que en cada iteración el algoritmo de la “burbuja” hace más intercambios que el “Sort”.

Veamos la solución:

```
import java.util.Scanner;

public class Burbuja {
    // Esta variable la usamos para pulsar intro y continuar
    static Scanner sc=new Scanner(System.in);

    // Generamos un array con números aleatorios entre el valor
```

```
// inferior y superior
public static void arrayAleatorio(int a[],int inf,int sup){
	int pos;
	int tam=a.length;
	for (pos=0;pos<tam;pos++){
		a[pos]=(int)(Math.random()*(sup-inf))+inf;
	}
}
// Método para intercambiar los valores en el array
public static void intercambio(int array[],int pos1,int pos2){
	int temp=array[pos1];
	array[pos1]=array[pos2];
	array[pos2]=temp;
}
// Método para mostrar los valores por pantalla
public static void mostrar(int a[]){
	for(int i=0;i<a.length;i++){
		System.out.print(a[i]+", ");
	}
	System.out.println();
}
// Algoritmo de ordenación BURBUJA
public static void burbuja(int a[], boolean mostrar){
	int i,j;
	long tiempoInicial;
	boolean ordenado=false;

	System.out.print("BURBUJA: ");
	mostrar(a);
	// Iniciamos el tiempo
	tiempoInicial=System.currentTimeMillis();
	// Acaba cuando está ordenado o hayamos recorrido todo
	for(i=0;ordenado==false && i<a.length;i++){
		// Suponemos que antes de iniciar una vuelta
		// está ordenado
		ordenado=true;
		// Recorremos el array intercambiando los menores
		for(j=1;j<a.length-i;j++){
			if(a[j]<a[j-1]){
				// Intercambio
				intercambio(a,j-1,j);
				ordenado=false;
			}
		}

		// Mostramos los valores
		if(mostrar==true){
			System.out.print("ITERACION "+(i+1)+":");
```

```
                    mostrar(a);
                }
            }
            System.out.print("Resuelto en:"+i+" iteraciones y "+
                ((System.currentTimeMillis()-tiempoInicial)/(float)1000)+
                 " segundos -> ");
            mostrar(a);
      }

      public static void main(String[] args) {
            int arrayEjemplo[]={7,4,5,-1,1};
            int arrayExtremo[]=new int[50000];
            arrayAleatorio(arrayExtremo,-30000,30000);
            burbuja(arrayEjemplo,true);
            System.out.print("Pulsa intro para seguir...");
            sc.nextLine();
            burbuja(arrayExtremo,false);
      }
}
```

Solución Auto-evaluación A5.13:

```
public class BusquedaLineal {
      // Método para mostrar los valores por pantalla
      public static void mostrar(int a[]){
            for(int i=0;i<a.length;i++){
                  System.out.print(a[i]+", ");
            }
            System.out.println();
      }
      // Algoritmo de búsqueda LINEAL
      public static int busquedaLineal(int a[], int elem){
            int pos=0;
            boolean encontrado=false;

            System.out.print("BUSQUEDA LINEAL DE "+elem+": ");
            // Acaba cuando encuentra el elemento o llegamos al final
            while(pos<a.length && encontrado==false){
                  if(a[pos]==elem){
                        encontrado=true;
                  }else{
                        pos++;
                  }
            }
            if(encontrado==true){
                  System.out.print("Resuelto en:"+pos+
                        " pasos -> Encontrado en la posición:"+
                        pos+" valor:"+a[pos]+"="+elem);
                  return pos;
```

```
            }else{
                  System.out.println("No se ha encontrado el valor.");
                  return -1;
            }
      }

      public static void main(String[] args) {
            int arrayEjemplo[]={-30,-19,-21,-16,27,14,-6,8,20,21,-6,
                                    -19,20,-23,22,-13,25,24,9,-27};
            int posEncontrado;

            posEncontrado=busquedaLineal(arrayEjemplo,-21);
            posEncontrado=busquedaLineal(arrayEjemplo,9);
      }
}
```

Solución Auto-evaluación A5.14:

```
public class BusquedaBinaria {
      // Método para mostrar los valores por pantalla
      public static void mostrar(int a[]){
            for(int i=0;i<a.length;i++){
                  System.out.print(a[i]+", ");
            }
            System.out.println();
      }
      // Algoritmo de búsqueda BINARIA
      public static int busquedaBinaria(int a[], int elem){
            int pasos=0,inf=0,sup=a.length-1,centro=0;
            boolean encontrado=false;

            System.out.print("BUSQUEDA BINARIA DE "+elem+": ");
            // Acaba cuando encuentra el elemento o llegamos al final
            while(inf<=sup && encontrado==false){
                  centro=((sup-inf)/2)+inf;
                  pasos++;
                  if(a[centro]==elem){
                        encontrado=true;
                  }else{
                        if(elem<a[centro]){
                              sup=centro-1;
                        }else{
                              inf=centro+1;
                        }
                  }
            }
            if(encontrado==true){
                  System.out.print("Resuelto en:"+pasos+
```

```
                        " pasos -> Encontrado en la posición:"+centro+
                        " valor:"+a[centro]+"="+elem);
                return centro;
            }else{
                System.out.println("No se ha encontrado el valor.");
                return -1;
            }
        }

        public static void main(String[] args) {
            int arrayEjemplo[]={-30,-27,-23,-21,-19,-19,-16,-13,-6,
                                -6,8,9,14,20,20,21,22,24,25,27};
            int posEncontrado;
            posEncontrado=busquedaBinaria(arrayEjemplo,-21);
            posEncontrado=busquedaBinaria(arrayEjemplo,9);
        }
}
```

Solución Auto-evaluación A5.15:

```
public class Autoeval515 {
    public static void sumaFilas(int a[][]){
        int f,c;
        int tamFilas=a.length;
        int tamCols=a[0].length;
        int filaSuma=0;
        for (f=0;f<tamFilas;f++){
            for(c=0;c<tamCols;c++){
                System.out.print(a[f][c]+"+");
                filaSuma=filaSuma+a[f][c];
            }
            System.out.println(" = "+filaSuma);
            filaSuma=0;
        }
    }

    public static void main(String[] args) {
        int a1[][]=
            {
                    {2,8,4,1,7,9,3},
                    {4,1,2,5,7,4,1},
                    {2,8,4,1,7,9,3},
                    {4,1,2,5,7,4,1},
                    {7,2,11,54,4,1,1}
            };
        sumaFilas(a1);
    }
}
```

Solución Auto-evaluación A5.16:

```
public class Autoeval516 {
	public static void sumaColumnas(int a[][]){
		int f,c;
		int tamFilas=a.length;
		int tamCols=a[0].length;
		int filaSuma=0;
		for(c=0;c<tamCols;c++){
			for (f=0;f<tamFilas;f++){
				System.out.print(a[f][c]+"+");
				filaSuma=filaSuma+a[f][c];
			}
			System.out.println(" = "+filaSuma);
			filaSuma=0;
		}
	}

	public static void main(String[] args) {
		int a1[][]=
			{
				{2,8,4,1,7,9,3},
				{4,1,2,5,7,4,1},
				{2,8,4,1,7,9,3},
				{4,1,2,5,7,4,1},
				{7,2,11,54,4,1,1}
			};
		sumaColumnas(a1);
	}
}
```

Solución Auto-evaluación A5.17:

```
public class Autoeval517 {
	public static void mostrarFigura(char fi[][]){
		int f=0,c=0,z=0;
		int tamFilas=fi.length;
		int tamCols=fi[0].length;
		for(f=0;f<tamFilas;f++){
			for(c=0;c<tamCols;c++){
				System.out.print(fi[f][c]);
			}
			System.out.println();
		}
	}

	public static void dibujo1(char m[][]){
		int mitad=(int)(m[0].length/2);
		int f,c;
		int tamFilas=m.length;
		int tamCols=m[0].length;
```

```
            for (f=0;f<tamFilas;f++){
                  for(c=0;c<tamCols;c++){
                        if(c<mitad){
                              m[f][c]='X';
                        }else{
                              m[f][c]='O';
                        }
                  }
            }
      }

      public static void main(String[] args) {
            char mitad[][]=new char[7][11];
            dibujo1(mitad);
            mostrarFigura(mitad);
      }
}
```

Solución Auto-evaluación A5.18:

La clase “Pila.java” sería:

```
public class Pila {
      private int valores[];
      private int pos;

      public Pila(int cantidad){
            valores=new int[cantidad];
            pos=0;
      }

      public void insertar(int valor){
            if(pos<valores.length){
                  valores[pos]=valor;
                  pos++;
            }else{
                  System.out.println("La pila está llena.");
            }
      }

      public void eliminar(){
            if(pos>0){
                  pos--;
            }else{
                  System.out.println("No quedan elementos en la pila.");
            }
      }

      public void mostrar(){
            for(int i=0;i<pos;i++){
```

```
                System.out.println("Pos "+i+" Valor: "+valores[i]);
            }
            System.out.println("********************");
        }

        public static void main(String[] args){
            // Creo la pila de 7 elementos
            Pila p=new Pila(7);
            // Pruebo si puedo eliminar
            p.eliminar();
            // Inserto los elementos
            p.insertar(3);
            p.insertar(5);
            p.insertar(2);
            p.insertar(1);
            p.mostrar();
            p.eliminar();
            p.eliminar();
            p.mostrar();
            p.insertar(10);
            p.mostrar();
            // Pruebo a insertar elementos hasta el máximo
            p.insertar(8);
            p.insertar(4);
            p.insertar(71);
            p.insertar(1);
            p.insertar(59);
            p.insertar(5);
            p.mostrar();
        }
}
```

Solución Auto-evaluación A5.19:

La clase “Cola.java” sería:

```
public class Cola {
    private int valores[];
    private int pos;

    public Cola(int cantidad){
        valores=new int[cantidad];
        pos=0;
    }

    public void insertar(int valor){
        if(pos<valores.length){
            valores[pos]=valor;
            pos++;
        }else{
```

```
                System.out.println("La cola está llena.");
            }
        }

        public void eliminar(){
            if(pos>0){
                for(int i=1;i<pos;i++){
                    valores[i-1]=valores[i];
                }
                pos--;
            }else{
                System.out.println("No quedan elementos en la Cola.");
            }
        }

        public void mostrar(){
            for(int i=0;i<pos;i++){
                System.out.println("Pos "+i+" Valor: "+valores[i]);
            }
            System.out.println("********************");
        }

        public static void main(String[] args){
            // Creo la cola de 7 elementos
            Cola c=new Cola(7);
            // Pruebo si puedo eliminar
            c.eliminar();
            // Inserto los elementos
            c.insertar(3);
            c.insertar(5);
            c.insertar(2);
            c.insertar(1);
            c.mostrar();
            c.eliminar();
            c.eliminar();
            c.mostrar();
            c.insertar(10);
            c.mostrar();
            // Pruebo a insertar elementos hasta el máximo
            c.insertar(8);
            c.insertar(4);
            c.insertar(71);
            c.insertar(1);
            c.insertar(59);
            c.insertar(5);
            c.mostrar();
        }
    }
```

Solución Auto-evaluación A5.20:

La clase "PilaDinamica.java" sería:

```
import java.util.Iterator;
import java.util.LinkedList;

public class PilaDinamica {
	private LinkedList<Integer> valores;

	public PilaDinamica(){
		valores=new LinkedList<Integer>();
	}

	public void insertar(int valor){
		valores.addLast(valor);
	}

	public Integer eliminar(){
		return valores.removeLast();
	}

	public void mostrar(){
		int pos=0;
		Iterator it=valores.iterator();
		while(it.hasNext()){
			System.out.println("Posición "+pos+": "+it.next());
			pos++;
		}
		System.out.println("**********************");
	}
	public static void main(String[] args) {
		PilaDinamica p=new PilaDinamica();
		// Inserto los elementos
		p.insertar(3);
		p.insertar(5);
		p.insertar(2);
		p.insertar(1);
		p.mostrar();
		p.eliminar();
		p.eliminar();
		p.mostrar();
		p.insertar(10);
		p.mostrar();
	}
}
```

La clase "ColaDinamica.java" sería:

```
import java.util.Iterator;
import java.util.LinkedList;
```

```
public class ColaDinamica {
	private LinkedList<Integer> valores;

	public ColaDinamica(){
		valores=new LinkedList<Integer>();
	}

	public void insertar(int valor){
		valores.addLast(valor);
	}

	public Integer eliminar(){
		return valores.removeFirst();
	}

	public void mostrar(){
		int pos=0;
		Iterator it=valores.iterator();
		while(it.hasNext()){
			System.out.println("Posición "+pos+": "+it.next());
			pos++;
		}
		System.out.println("**********************");
	}
	public static void main(String[] args) {
		ColaDinamica c=new ColaDinamica();
		// Inserto los elementos
		c.insertar(3);
		c.insertar(5);
		c.insertar(2);
		c.insertar(1);
		c.mostrar();
		c.eliminar();
		c.eliminar();
		c.mostrar();
		c.insertar(10);
		c.mostrar();
	}
}
```