6
Estructuras de almacenamiento

Índice:

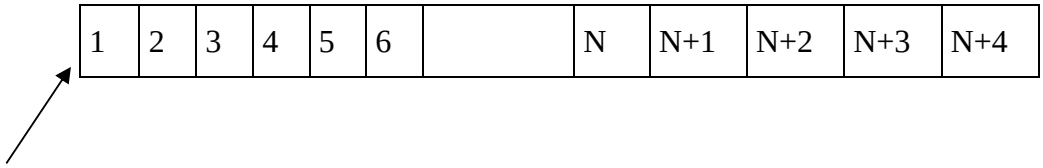
1. Arrays o vectores

Un **array** es una estructura de datos incorporada la mayoría de lenguajes de programación que está formado por una colección finita de elementos homogéneos (del mismo tipo) y ordenados (hay un primer elemento, un segundo elemento, etc.) que se referencian bajo un nombre común. Así definida, esta colección de datos forma una estructura estática, es decir, que su tamaño será conocido en tiempo de compilación.

En Java se pueden definir vectores o arrays de varios tipos (bolean, int, byte, etc.) y también arrays de objetos. De esta manera se pueden almacenar varios valores en cada posición de memoria.

Un array se compone de una serie de posiciones consecutivas en memoria.

MEMORIA



A los vectores se accede mediante un subíndice, si nuestro vector se llama Temperaturas y se quiere acceder a la posición N, se tendrá que escribir Temperaturas[N] en el programa para obtener la información de esa posición de memoria. N puede ser una variable o bien un valor concreto.

1.1. Declaración de vectores

En Java se pueden declarar vectores de dos formas diferentes. Nuestro vector de temperaturas, se puede declarar de las siguientes formas:

```
byte[ ] Temperaturas;
byte Temperaturas[ ];
```

En estas declaraciones no se define el tamaño del vector, solo se ha especificado el tipo de elementos que va a contener dicho vector.

1.2. Creación de vectores

Java trata a los vectores como si fuesen objetos, por lo tanto la creación del vector Temperaturas será del siguiente modo:

```
Temperaturas[ ] = new byte[100];
```

Lo que implica reservar en memoria 100 posiciones de tipo byte. En los vectores cuando se reservan N posiciones de memoria, los datos se almacenan en las posiciones 0, 1, 2…, N-1.

El tamaño también puede asignarse mediante una variable de la siguiente forma:

```
int v = 100;
```

```
byte[ ] Temperaturas;
Temperaturas[ ] = new byte[v];
```

O también de la siguiente forma (fundiendo las líneas 2 y 3 en una sola):

```
int v = 100;
byte[ ] Temperaturas = new byte[v];
```

1.3. Inicialización de vectores

Si no se especifica ningún valor, los elementos de un vector se inicializan a: 0 si son numéricos, *null* si son objetos, *false* si son bolean y *'\u000'* si son caracteres.

También es posible inicializarlos específicamente de la siguiente forma:

```
byte[ ] Temperaturas={10, 11, 12, 11, 10, 9, 18, 19, 14, 13,15, 15};
```

En este código se ha creado un vector de 12 posiciones del tipo byte con los valores especificados.

1.4. Métodos de los vectores

Java maneja los vectores como objetos, por tanto hay una serie de métodos heredados de la clase Object que está en el paquete Jva.lang:

- **equals**. Permite discernir si dos referencias son el mismo objeto.
- **clone**. Duplica un objeto.

Un ejemplo de utilización de estos métodos es el siguiente:

```
byte[ ] Temperaturas1 = {10, 11, 12, 11, 10, 9, 18, 19, 14, 13,15, 15};
byte[ ] Temperaturas2 = (byte[ ]) Temperaturas1.clone;
byte[ ] Temperaturas3 = Temperaturas1;
if (Temperaturas1.equals(Temperaturas2))
        System.out.println("Temperaturas1 == Temperaturas2");
else
        System.out.println("Temperaturas1 != Temperaturas2");
if (Temperaturas1.equals(Temperaturas3))
        System.out.println("Temperaturas1 == Temperaturas3");
else
        System.out.println("Temperaturas1 != Temperaturas3");
```

El programa mostrará lo siguiente: "Temperaturas1 != Temperaturas2" y "Temperaturas1 == Temperaturas3". En el primer caso aunque los datos son los mismos, el objeto es diferente el objeto es diferente y en el segundo caso, al asignar Temperaturas1 == Temperaturas3 hace que la segunda referencia apunte al mismo lugar de memoria que la primera y no se dupliquen los datos. En ese caso el método *equals* si da como resultado *true*.

1.5. Utilización de los vectores

Un ejemplo de utilización de los vectores se ve en el siguiente programa:

```
public class temperaturas {
        private static int[ ] temperaturas1;
        final static int POS = 10;                  //número de posiciones del vector
        public static void main(String[ ] args) {
                int dato = 0; media = 0;
                temperaturas1 = new int[POS];
                for (int i=0; i<POS; i++) {      // leer los valores del vector
                        try {
                                System.out.println("Introduzca temperature: ");
                                String sdato = System.console( ). readLine( );
                                dato = Integer.parseInt(sdato);
                        } catch(Exception e) {
                                System.out.println("Error al introducir datos");
                        }
                        temperaturas1[i] = dato;
                }
                for (int i=0; i<POS; i++) {      // hacer la media
                        media = media + temperaturas1[i];
                }
                media = media / POS;
                System.out.println("La media de temperaturas es: " + media);
        }
}
```

En este programa se leen 10 temperaturas de diferentes ciudades y luego se muestra por pantalla la media de temperaturas. La constante POS contiene el número de temperaturas a registrar, que en el ejemplo vale 10 pero se puede aumentar o disminuir su valor y el programa seguirá funcionando.

2. Arrays multidimensionales o matrices

La definición de matrices es similar a la de vectores aumentando el número de dimensiones. Por ejemplo, una matriz de enteros de dos dimensiones con 5 filas y 8 columnas se creará de la siguiente manera:

```
int [ ][ ] matriz = new int [5][8];
```

La inicialización de la matriz en el momento de su definición se hará del siguiente modo:

```
int [ ][ ] matriz = { {1, 4, 5},{6, 2, 5,} };
```

En el código anterior se ha creado un array de 2 filas y 3 columnas.

```
System.out.println("La matriz tiene por filas el valor: " + matriz.length);
System.out.println("La media tiene por columnas el valor: " + matriz[0].length);
```

El código anterior muestra en la primera línea el número de filas de la matriz (2) y la segunda línea el número de columnas de la fila 0 de la matriz (3).

	COLUMNAS		
FILAS	matriz[0][0]	matriz[0][1]	matriz[0][2]
	matriz[1][0]	matriz[1][1]	matriz[1][2]

El acceso a la matriz se hace igual que cuando se trabaja con vectores (matriz[fila][columna]). Veamos como podemos almacenar en cada celda de la matriz la suma de la posición de la fila y la columna:

```
for (int i = 0; i < 2; i++)  {
        for (int j = 0; j < 3; j++)  {
                matriz [i] [j] = I +j;
        }
}
```

3. Cadenas de caracteres

Las cadenas de caracteres en java se tratan como objetos de la clase String.

Una cadena de caracteres es un vector de elementos de tipo *char*.

```
char[] [] nombre1 = { ‘p’, ‘e’, ‘p’, ‘e’};
char[] [] nombre2 = { 112, 102, 112, 101};
char[] [] nombre3 = new char[4];
```

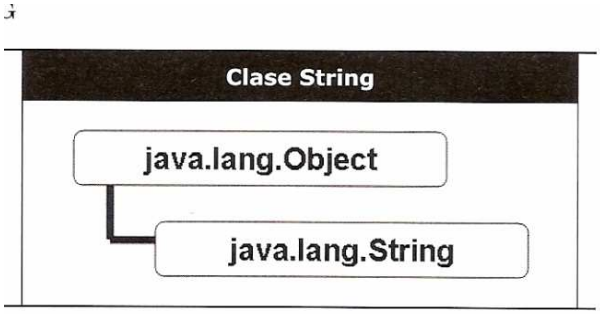
En el código anterior las variables nombre1 y nombre2 contienen exactamente lo mismo ya que java almacena los caracteres con sus códigos ASCII correspondientes (a la ‘p’ le corresponde el 112 y a la ‘e’ el 101). La variable nombre3 se ha creado como una cadena de 4 caracteres pero todavía no se ha inicializado y, por tanto, sus 4 posiciones contendrán el valor ‘\0’.

La siguiente línea de código muestra la longitud del string “HOLA”:

```
System.out.println(“HOLA“.length());
```

3.1. La clase String

La clase *String* pertenece al paquete **java.lang** y proporciona todo tipo de operaciones con cadenas de caracteres. Esta clase ofrece métodos de conversión a cadena de números, conversión a mayúsculas, minúsculas, reemplazamiento, concatenación, comparación, etc.



Veamos los métodos más importantes de la clase *String*:

- **String (String dato).** Constructor de la clase *String*.

```
String cad1 = “Pepe”;
String cad1 = new String(“Lionel”);
String cad1 = new String(cad2);
```

Creamos tres objetos de la clase *String*, el objeto *cad3* se crea a partir del objeto

cad2 y contendrá los mismos datos “Lionel”.

- **int length( ).** Muestra la longitud de un objeto de la clase *String*.

```
String cad1 = “CHELO”;
System.out.println(“HOLA“.length());
```

Muestra la longitud del objeto String cad1 (5).

- **String concat (String s).** Devuelve un objeto de la clase *String* que es la concatenación de dos Strings.

```
String cad1 = “Andy”;
cad1 = cad1.concat(“ Rosique”);
System.out.println(“HOLA“.length());
```

Muestra en pantalla la concatenación “Andy Rosique”.

- **String toString ( ).** Devuelve el propio *String*.

```
String cad1 = “Emilio”;
String cad2 = “ Anaya”;
System.out.println(cad1.toString( ) + cad2.toString( ));
```

Muestra en pantalla la concatenación “Emilio Anaya”.

- **int compareTo (String s).** Compara el objeto *String* con el *String* pasado como parámetro y devuelve un número:

  < 0 Si el *String* que la llama es **menor** al *String* que se le pasa como parámetro.

  = 0 Si el *String* que la llama es **igual** al *String* que se le pasa como parámetro.

  >0 Si el *String* que la llama es **mayor** al *String* que se le pasa como parámetro.

  El método va comparando letra a letra ambos *String* y si encuentra que una letra u otra es mayor o menor que otra deja de comparar.

```
String cad1 = “EMMA”;
String cad2 = “MARIA”;
System.out.println(cad1.compareTo(“emma”);                  //muestra -32
System.out.println(cad1.compareTo(“EMMA”);                  //muestra 0
System.out.println(cad1.compareTo(“EMMA MORENO”);           //muestra -32
System.out.println(cad2.compareTo(“MARIA AMPARO”);          //muestra -7
System.out.println(cad2.compareTo(“MAREA”);                 //muestra 4
```

- **boolean equals ( ).** Compara el contenido de dos objetos del tipo *String*.

```
String cad1 = “EMMA”;
String cad2 = new String(“EMMA”);
if (cad1.equals (cad2))  System.out.println(“SON IGUALES”);
else  System.out.println(“SON DIFERENTES”);
```

Muestra en pantalla la cadena “SON IGUALES”.

- **String toLowerCase ( ).** Convierte las letras mayúsculas del objeto *String* a minúsculas.

```
String cad1 =  “PEDRO ruiz”);
String cad2 = cad1.toLowerCase( );
System.out.println(cad2.toString());
```

Muestra en pantalla la cadena “pedro ruiz”.

- **String toUpperCase ( ).** Convierte las letras minúsculas del objeto *String* a mayúsculas.

```
String cad1 =  "JUAn serrano");
String cad2 = cad1.toUpperCase( );
System.out.println(cad2.toString());
```

Muestra en pantalla la cadena "JUAN SERRANO".

- **String trim ( ).** Elimina los espacios en blanco que contenga el objeto *String* al principio y al final del mismo.

```
String cad1 =  "        MAYKA        ");
String cad2 = cad1.trim( );
System.out.println(cad2.toString());
```

Muestra en pantalla la cadena "MAYKA".

- **String replace (char car, char newcar).** Reemplaza cada ocurrencia del carácter *car* por el carácter *newcar*.

```
String cad1 =  "JUAN SUAREZ");
String cad2 = cad1.replace('U', 'O');
System.out.println(cad2.toString());
```

Muestra en pantalla la cadena "JOAN SOAREZ".

- **String substring (int ini, int fin).** El método devuelve un nuevo *String* que será la subcadena que comienza en el carácter ini y termina en el carácter fin (que no se muestra). Si no se especifica el segundo parámetro, devolverá hasta el final de la cadena.

```
String cad1 =  "JUAN CARLOS MORENO");
System.out.println(cad1.substring(5, 11));
System.out.println(cad1.substring(12));
```

Muestra en pantalla las cadenas "CARLOS" y "MORENO".

- **boolean startsWith (String cad).** El método devuelve **true** si el objeto *String* comienza con la cadena *cad*, en caso contrario devuelve **false**.

```
String cad1 =  "MAYKA MORENO");
System.out.println(cad1.startsWith("JUAN"));
System.out.println(cad1.startsWith("MAY"));
```

Muestra en pantalla *false* y *true*.

- **boolean endsWith (String cad).** El método devuelve **true** si el objeto *String* termina con la cadena *cad*, en caso contrario devuelve **false**.

```
String cad1 =  "MARIA AMPARO");
System.out.println(cad1.endsWith("paro"));
System.out.println(cad1.endsWith("PARO"));
System.out.println(cad1.endsWith("ARIA"));
```

Muestra en pantalla *false*, *false* y *true*.

- **char charAt (int pos).** El método devuelve el carácter de la posición **pos** del objeto *String*. Si el **pos** no está entre 0 y lenght( )-1, Java lanzará una excepción.

```
String cad1 =  "AMPARO HEREDIA";
System.out.println(cad1.charAt(0)+ "  "+cad1.charAt(7));
System.out.println(cad1. startsWith("MAY"));
```

Muestra en pantalla la cadena “A H”.

- **int indexOf(int c)** o **int indexOf(String s).** El método admite dos tipos de parámetros y nos permite encontrar la primera ocurrencia de un carácter o una subcadena dentro de un objeto *String*. En caso de no encontrarlo, el método devuelve el valor -1.

```
String cad1 =  “EMMA MORENO;
System.out.println(cad1.indexOf(‘M’));
System.out.println(cad1.indexOf(‘J));
System.out.println(cad1.indexOf(“MO”));
System.out.println(cad1.indexOf(“MI”));
```

Muestra en pantalla el siguiente resultado: 1, -1. 5 y -1.

- **char [ ] toChar (int pos).** El método devuelve un vector o array de caracteres a partir del propio objeto *String*.

```
String cad1 =  “LORO FELIPE”);
char cad2 [ ] = cad1.toChar( );
```

El código anterior crea un array de caracteres *cad2* que contiene “LORO FELIPE”.

- **String valueOf (int dato).** Convierte un número a un objeto *String*. La clase *String* es capaz de convertir los tipos primitivos *int*, *long*, *float* y *double*.

```
int edad1 = 6;
String str  = String.valueOf(edad1);
float edad2 = 6;
str  = String.valueOf(edad2);
long edad1 = 6;
str  = String.valueOf(edad3);
double edad2 = 6.5;
str  = String.valueOf(edad4);
```

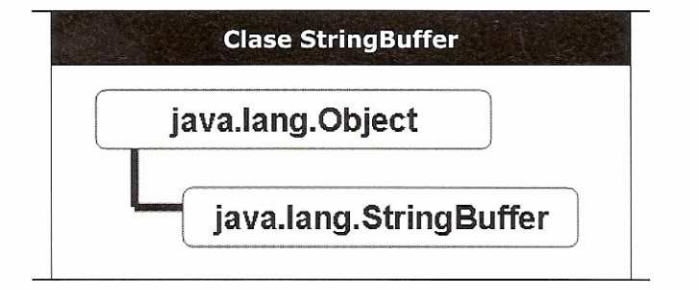
Para convertir un String en un número se haría de la siguiente forma:

```
String snumero = “     6      “;
int numero=Integer.parseInt(snumero); // error, no se han eliminado los blancos
int numero=Integer.parseInt(snumero.trim( ));

//para números con decimales se usará el siguiente código
String snumero = “     6.5      “;
double numero = Double.valueOf(snumero).doubleValue( );
```

3.2. La clase StringBuffer

Los objetos de la clase String NO son modificables sino que los métodos que actúan sobre los objetos devuelven un objeto nuevo con las modificaciones realizadas. En cambio, los objetos StringBuffer SÍ son modificables.



- **StringBuffer ([arg]).** Constructor de la clase *StringBuffer*.

```
StringBuffer nombre = new StringBuffer(”Pepe”);
StringBuffer nombre = new StringBuffer(80);
StringBuffer nombre = new StringBuffer( );
```

La segunda línea del código crea un objeto vacío con una capacidad de 80 caracteres. En la tercera línea de código no se especifica la capacidad pero por defecto la deja a 16.

- **int length( ).** Muestra la longitud del objeto *StringBuffer* .

```
StringBuffer nombre = new StringBuffer (“Pepe”);
System.out.println(nombre.length());
```

La salida por pantalla del código anterior será 4.

- **int capacity ( ).** Muestra la capacidad del objeto del tipo *StringBuffer*.

```
StringBuffer nombre = new StringBuffer(”Pepe”);
System.out.println(nombre.capacity( ));
```

La salida por pantalla del código anterior será 20 aunque la longitud es de 4. eso es debido a que cuando se crea el objeto Java le otorga una capacidad igual al número de caracteres almacenados más 16.

- **StringBuffer append (argumento).** Añade el argumento al final de la cadena de caracteres *StringBuffer*. El tipo de argumento puede ser *int*, *long*, *float*, *double*, *boolean*, *char*, *char[ ]*, *String* y *Object*.

```
StringBuffer nombre = new StringBuffer(”Juan Carlos”);
StringBuffer apellidos = new StringBuffer(” Moreno Pérez”);
nombre.append(apellidos);
System.out.println(nombre);
```

La salida por pantalla del código anterior será “Juan Carlos Moreno Pérez”

- **StringBuffer insert (int pos, arg).** Añade el argumento en la posición **pos** de la cadena de caracteres *StringBuffer*. El tipo de argumento puede ser *int*, *long*, *float*, *double*, *boolean*, *char*, *char[ ]*, *String* y *Object*.

```
StringBuffer nombre = new StringBuffer(”EMMA”);
StringBuffer apellidos = new StringBuffer(” MORENO”);
nombre.insert(nombre.length( ), apellidos);
System.out.println(nombre);
```

La salida por pantalla del código anterior será “EMMA MORENO”

- **StringBuffer reverse ( ).** Invierte la cadena de caracteres que contiene

```
StringBuffer nombre = new StringBuffer(”TURRION”);
nombre.reverse( );
System.out.println(nombre);
```

La salida por pantalla del código anterior será “NOIRRUT.

- **StringBuffer delete (int x, int y).** Elimina los caracteres entre las posiciones x e y del objeto *StringBuffer*.

```
StringBuffer nombre = new StringBuffer(”RAUL JESUS TURRION”);
nombre.insert(nombre.delete(5, 10));
System.out.println(nombre);
```

La salida por pantalla del código anterior será “RAUL TURRION”

- **StringBuffer replace (int x, int y, String s).** Reemplaza los caracteres entre las posiciones x e y por el *String* s del objeto *StringBuffer*.

```
StringBuffer nombre = new StringBuffer(”RAUL JESUS”);
nombre.insert(nombre.replace(5, 10, ”TURRION”));
System.out.println(nombre);
```

La salida por pantalla del código anterior será “RAUL TURRION”

- **String substring (int x, int y).** Devuelve un *String* que contiene la cadena que comienza en el carácter x hasta el carácter y-1 (o hasta el final de la cadena si no se especifica y).

```
StringBuffer nombre = new StringBuffer(“RAUL JESUS TURRION”);
String turri = nombre.substring (0, 4) + nombre.substring (10);
System.out.println(turri);
```

El código anterior muestra en pantalla la cadena “RAUL TURRION”

- **String toString ( ).** Devuelve un objeto *String* el cual es una copia del objeto *StringBuffer*.

```
StringBuffer nombre = new StringBuffer (“TURRION”);
String turri = nombre.toString( );
System.out.println(turri);
```

El código anterior crea un objeto *String* a partir de un objeto *StringBuffer* y muestra su contenido en pantalla “TURRION”.

- **char charAt (int x).** Devuelve el carácter que está en la posición **x** del objeto *StringBuffer*.

```
StringBuffer nombre = new StringBuffer (“EMMA”);
System.out.println(nombre.charAt(0));
```

El código anterior muestra por pantalla el carácter ‘E’.

- **void setCharAt (int x, char c).** Reemplaza el carácter que está en la posición **x** del objeto *StringBuffer* por el carácter **c**.

```
StringBuffer nombre = new StringBuffer (“EMMA”);
nombre.setCharAt(0, ‘e’ );
System.out.println(nombre.toString( ));
```

El código muestra por pantalla la cadena de caracteres “eMMA”.

3.3. La clase StringTokenizer

Esta clase permite dividir una cadena de caracteres en elementos independientes si estos están separados por un espacio en blanco, retorno de carro (\r), retorno de línea (\n), avance de página(\f) o un tabulador (\t).

Veamos un ejemplo de utilización:

```
StringTokenizer str;
Str = new StringTokenizer(”UNO DOS TRES JUAN PERICO Y_ANDRES”);
System.out.println(“La cadena str tiene “+str.countTokens( ) +” elementos”);
while (str.hasMoreTokens( ))
        System.out.println(“Elemento: “+str.nextToken( ));
```

Este código mostrará que la cadena str tiene 6 elementos y los mostrará en pantalla

uno a uno.

Para usar esta clase se debe importar la clase StringTokenizer o todas las clases del paquete java.util.

```
import java.util.StringTokenizer;          // o import java.util.*;
```

También es posible especificar los delimitadores dentro del constructor de la clase, veamos un ejemplo que usa como delimitador '|':

```
str = new StringTokenizer("UNO|DOS|TRES|JUAN|PERICO|Y_ANDRES");
```

4. Arrays o vectores de objetos String

Un vector de objetos String permite en un único vector almacenar diferentes elementos que son todos de tipo String. Veamos un ejemplo en el que se leerán nombres por teclado y se irán almacenando en un vector de Strip. A continuación se mostrarán por pantalla dichos nombres según la posición que se han leído:

```
import java.io.*;
public class test {
  private static String[ ] lista;
  final static int POS = 10;       //número de posiciones del array
  public static void muestra( )  {
            for (int i = 0; i < POS; i++)
                      System.out.println("Elemento "+ i+" "+lista[i]);
            }
  public static void main(String[ ] args) throws IOException{
            lista = new String[POS];
            String ln="";
            BufferedReader entrada = new BufferedReader(new InputStreamReader
            (System.in));
            for (int i = 0; i < POS; i++)  {
                      System.out.println ("Introduce un nombre: ");
                      lista[i] = entrada.readLine();
            }
            System.out.println("");
            muestra( );
            System.out.println("");
  }
}
```

5. Algoritmos de ordenación

Los algoritmos de ordenación se usan para organizar los datos en un array. Hay diferentes algoritmos para ordenar vectores, algunos son muy rápidos cuando el vector está casi ordenado, otros lo son cuando está muy desordenado y a otros no les afecta en gran medida.

5.1. Ordenación por el método de la burbuja

Es un algoritmo muy utilizado porque es sencillo y fácil de entender. En cada iteración se pone el elemento más pequeño no ordenado en su lugar correcto (al igual que las burbujas de aire en el agua salen a la superficie, los elementos menores se van colocando al principio del vector de forma ordenada). También podría implementarse colocando el elemento mayor en su lugar correcto conociéndose como el algoritmo de la

plomada.

Explicación del algoritmo:

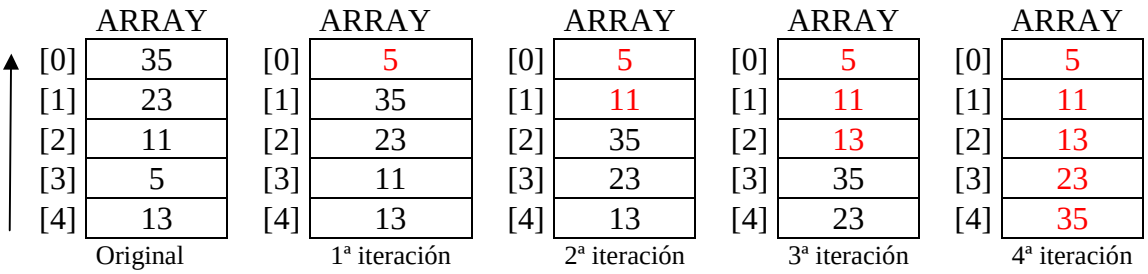
Se trata de recorrer el array repetidas veces, de forma que cada iteración ponga el elemento menor NO ordenado en su sitio, lo cual provocará cambios en la posición de otros elementos del array. Se comienza con el elemento de la posición N-ésima y se van comparando sucesivos pares de elementos, intercambiándolos si el elemento de abajo del par (el situado en la casilla de índice más alto) es más pequeño que el elemento que le precede (situado en la casilla de índice más bajo). De esta forma, el elemento menor va "burbujeando" hacia arriba hasta el tope del array. Así, en la primera iteración subirá a la 1ª posición el elemento menor. En la segunda iteración, usando la misma técnica, el elemento menor de la parte no ordenada del array subirá a la 2ª posición y así sucesivamente con los elementos restantes del array.

Se utilizan dos índices. El índice i separa la parte ordenada de la no ordenada, señalando siempre el primer elemento de la parte no ordenada, mientras que el índice j se mueve por la parte no ordenada del array buscando un elemento para, según las especificaciones, ponerlo en su lugar. Los elementos iguales quedarán en posiciones consecutivas.

```
public class test {
	private int[] array;

	public void cargar() {
		array=new int[10];
		for(int f=0;f<10;f++) {
			array[f]=(int)(Math.random()*100);}
	}
	public void burbuja() {
		for (int i = 0; i< 9; i++)
			for (int j = 9; j > i; j--)
				if (array[j] < array[j-1]) {
					int aux = array[j];
					array[j] = array [j-1];
					array[j-1] = aux;
				}
			}
		}
	}
	public void imprimir() {
		System.out.println("Vector ordenado por método burbuja.");
		for(int f=0;f<array.length;f++) {
			System.out.println(array[f]);
		}
	}
	public static void main(String[] ar) {
	test pv=new test();
	pv.cargar();
	pv.burbuja();
	pv.imprimir();
	}
}
```

Ejemplo: Suponiendo un array de enteros con los datos que aparecen en el gráfico siguiente, se aplicará el algoritmo descrito anteriormente sobre el array que aparece como original en la figura.



5.2. Ordenación por el método de selección directa

Explicación del algoritmo:

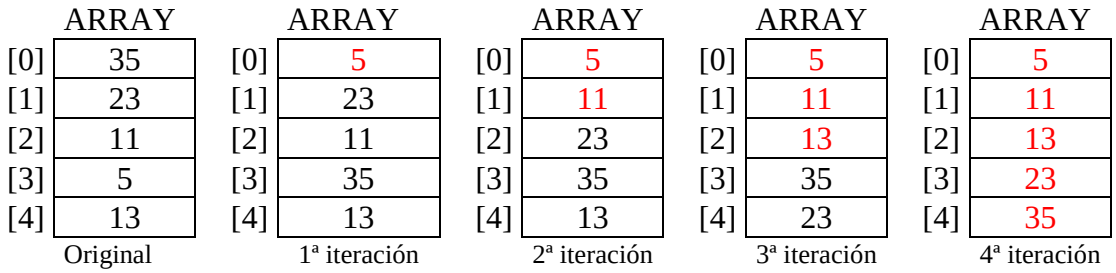
El método divide el array en una parte ordenada y otra por ordenar. En principio, la parte no ordenada es todo el array. Se recorre el array repetidas veces, de forma que, en cada iteración, se busca, en la parte no ordenada, el elemento más pequeño para colocarlo en su lugar.

El método se conoce también como método de "búsqueda del menor" si se realiza la ordenación de forma ascendente o "búsqueda del mayor" si se realiza de forma descendente.

Veamos el algoritmo: (solo el método de ordenación, el programa ya está hecho antes)

```
public void seleccion( ) {
        int minimo, aux;
        for (i = 0; i< N; i++) {
                minimo = i;
                for (j = i+1; j < N; j++)
                        if (array[j] < array[minimo])
                                minimo = j;
                        aux = array[minimo];
                        array[minimo] = array [i];
                        array[i] = aux;
        }
```

Ejemplo: Con el mismo array del ejemplo anterior, seguimos la traza para el algoritmo descrito. Observe que, en cada iteración, el elemento más pequeño de la parte no ordenada se coloca tras el último elemento de la parte ordenada.



5.3. Ordenación por el método de inserción directa

Explicación del algoritmo:

El método divide el array en una parte ordenada y otra por ordenar, tomando el primer elemento de la parte no ordenada y haciéndolo avanzar hasta colocarlo en su lugar correspondiente dentro de la parte ordenada. Es decir, comienza considerando que el primer elemento está ordenado, continúa ordenando el segundo elemento respecto al primero, es decir, o se coloca delante o se queda donde está. Se hace avanzar el índice que señala el elemento no ordenado. Se toma el tercero y se coloca en su lugar correcto respecto a los dos anteriores. Se hace avanzar el índice y se repite el proceso hasta que todos los elementos ocupen su lugar correcto, con lo cual el array quedará ordenado según las especificaciones de partida.

Veamos el algoritmo:

```
public  void insercion ( ) {
        int aux;
        int j;
        for (int i=1; i<=9; i++) {
                aux = array[i];
                for (j=i-1; j>=0 && array[j]>aux; j--){
                        array[j+1] = array[j];
                        array[j] = aux;
                }
        }
}
```

Ejemplo: Con el mismo array del ejemplo anterior, seguimos la traza para el algoritmo descrito. Observe que, en cada caso se van tomando uno a uno los elementos de la parte no ordenada para buscarle su “hueco” en la parte ordenada.

	ARRAY		ARRAY		ARRAY		ARRAY		ARRAY
[0]	35	[0]	23	[0]	11	[0]	5	[0]	5
[1]	23	[1]	35	[1]	23	[1]	11	[1]	11
[2]	11	[2]	11	[2]	35	[2]	23	[2]	13
[3]	5	[3]	5	[3]	5	[3]	35	[3]	23
[4]	13	[4]	13	[4]	13	[4]	23	[4]	35
	Original		1ª iteración		2ª iteración		3ª iteración		4ª iteración

6 Búsqueda de datos dentro de un array

Una de las operaciones más frecuentes que se realizan en los arrays es la búsqueda de información dentro de ellos. La forma más fácil de buscar información en un array es realizar una búsqueda secuencial hasta encontrar el dato, pero no es muy eficiente pues se pierde mucho tiempo en la búsqueda en arrays muy grandes. Hay un método más eficiente llamado búsqueda binaria que funciona de la siguiente forma:

BÚSQUEDA BINARIA

La búsqueda binaria **sólo es aplicable a arrays ordenados**, consiste en dividir el

array en dos mitades y buscar el dato en la mitad que corresponda, siguiendo el mismo método hasta encontrarlo o hasta estar seguros que el elemento no se encuentra en dicho array, lo cual no implica recorrer el array entero. En principio, se compara el dato a buscar con el elemento que ocupa la posición mitad del array. Si el dato no coincide con este elemento, se reduce el intervalo de búsqueda a la mitad inferior o superior del array dependiendo de si el dato a buscar es mayor o menor que el elemento situado en la mitad del array. Este proceso se va repitiendo hasta encontrar el dato en el vector o hasta quedarse sin segmentos de búsqueda.

Este método es eficaz con arrays con muchos elementos ordenados, en arrays pequeños puede que sea menos eficiente que la búsqueda secuenciar porque realiza más comparaciones que aumentan el tiempo de ejecución.

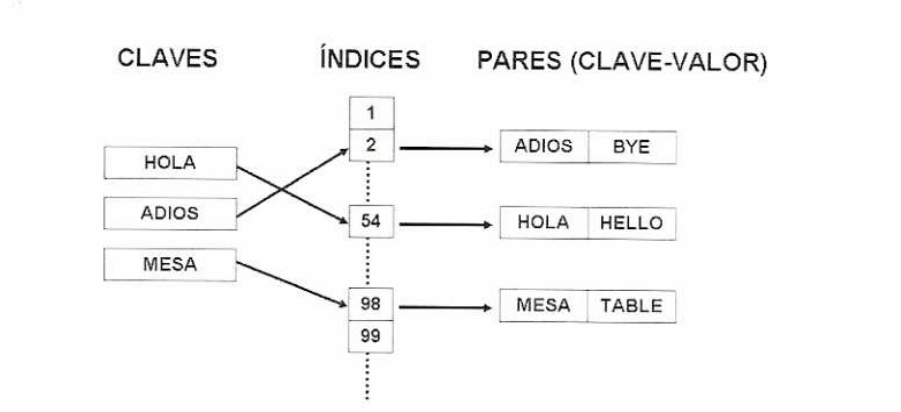
Veamos el algoritmo:

```
int Busqueda_Bin(int dato) {
        int inicio= 0, fin= 10-1, med = 10/2, encontrado=-1;

        while (inicio <= fin && encontrado == -1)  {
                if (array[med] == dato)
                        encontrado = med;
                        /* mover inicio y fin para dividir el área de búsqueda */
                else {
                        if (array[med] > dato)    // Dato estará del principio a la mitad
                                fin = med - 1;
                        else                      // Dato estará de la mitad al final
                                inicio = med + 1;
                        med = ( inicio + fin ) / 2;
                }
        }
        return encontrado;
}
```

7. La clase Hashtable

Esta clase se encuentra en el paquete **java.util**. Una Hashtable implementa una tabla hash la cual crea una especie de diccionario que relaciona claves con valores.



En el siguiente ejemplo comentado se muestra cómo se crea y se trabaja con una Hashtable como la de la siguiente figura:

```
import java.util.*;
public class diccionario {
        public static void main (String[ ] args) {
                Hashtable dic = new Hashtable( );
                dic.put("HOLA", "HELLO");
                dic.put("ADIOS", "BYE");
                dic.put("MESA", "TABLE");
                dic.put("SILLA", "CHAIR");
                dic.put("CABEZA", "HEAD");
                dic.put("CARA", "FACE");
                String saludo = (String) dic.get("HOLA");
                String despedida = (String) dic.get("ADIOS");
                String brazo = (String) dic.get("BRAZO");
                System.out.println("HOLA :"+saludo);    //muestra HELLO en pantalla
                System.out.println("ADIOS :"+despedida);   //muestra BYE en pantalla
                System.out.println("BRAZO :"+brazo);    //muestra null en pantalla
                System.out.println("dic contiene "+dic.size( )+ " pares.");
                if (dic.containsKey("HOLA"))
                        System.out.println("dic contiene HOLA como clave");
                else
                        System.out.println("dic NO contiene HOLA como clave");
                if (dic.contains("HELLO"))
                        System.out.println("dic contiene HELLO como valor");
                else
                        System.out.println("dic NO contiene HELLO como valor");
                System.out.println("Mostrando todos los valores de la tabla Hash….");
                Enumeration k = dic.keys( );
                while (k.hasMoreElements( ) )
                        System.out.println(k.nextElement( ));
        }
}
```