

El manifiesto del JAR creado anteriormente es muy simple, tan solo contendría los siguientes datos:

```
Manifest-Version: 1.0  
Created-By: 1.6.0_19 (Sun Microsystems Inc.)
```

Como se puede observar contiene la versión de Java con el que ha sido creado y poco más. En el caso de que utilicemos funcionalidades más avanzadas de la herramienta JAR el contenido de este fichero cambiaría sustancialmente.

7.3 CADENAS DE CARACTERES



Recuerda

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

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

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

En el código anterior las variables nombre1 y nombre2 contienen exactamente lo mismo dado que internamente Java almacena los caracteres con sus símbolos 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'.



Recuerda

Para comprobar que las cadenas de caracteres en Java se tratan como objetos de la clase *String* prueba a hacer lo siguiente:

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

La anterior línea muestra la longitud del string/cadena de caracteres que en este caso sería 4.

7.3.1 LA CLASE STRING

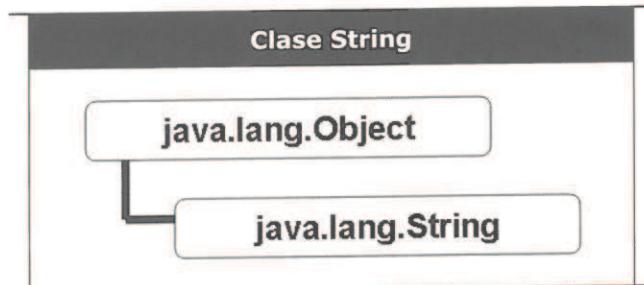


Figura 7.3. Jerarquía de la clase String

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.



Recuerda

Aparte de todos los siguientes métodos, el propio lenguaje Java ofrece el operador de concatenación `+`. Un ejemplo de utilización es:

```
System.out.println("Longitud de la cadena HOLA: "+ "HOLA".length());
```

■ `String(String dato)`. Constructor de la clase *String*

```
String cad1 = "Pepe";
String cad2 = new String("Lionel");
String cad3 = new String(cad2);
```

Las tres líneas de código anterior crean objetos de la clase *String*. Nótese como el objeto *cad3* está creado 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(cad1.length());
```

El código anterior muestra la longitud del objeto *string cad1* (5).

■ `String concat(String s)`. Devuelve un objeto fruto de la concatenación/unión de un objeto *String* con otro.

```
String cad1 = "Andy";
cad1=cad1.concat(" Rosique");
System.out.println(cad1);
```

El código anterior concatena las cadenas "Andy" y "Rosique", y muestra por pantalla el resultado de la concatenación ("Andy Rosique").

■ String `toString()`. Devuelve el propio *String*.

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

El código anterior aprovecha el operador concatenación “+” para mostrar por pantalla la cadena “Emilio Anaya”.

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

- < 0 Si es menor el *string* desde el que se hace la llamada al *String* pasado como parámetro.
- = 0 Si es igual el *string* desde el que se hace la llamada al *String* pasado como parámetro.
- > 0 Si es mayor el *string* desde el que se hace la llamada al *String* pasado 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"));
System.out.println(cad1.compareTo("EMMA"));
System.out.println(cad1.compareTo("EMMA MORENO"));
System.out.println(cad2.compareTo("MARIA AMPARO"));
System.out.println(cad2.compareTo("MAREA"));
```

El anterior código mostrará por pantalla los siguientes datos: -32, 0, -7, -7 y 4.

**iCuidado!**

El método `compareTo` distingue mayúsculas de minúsculas. Las mayúsculas están antes por orden alfabético que las minúsculas, por lo tanto 'A' es menor que 'a'.

■ boolean `equals()`. Este método sirve para comparar 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");
}
```

El código anterior mostrará por pantalla “SON IGUALES”.

A FONDO**DIFERENCIA ENTRE EL MÉTODO EQUALS Y EL OPERADOR ==**

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

El código anterior mostrará por pantalla las siguientes cadenas: "SON IGUALES", "SON DIFERENTES". En el primer caso los dos objetos son iguales porque contienen la misma secuencia de caracteres y el segundo de los casos son diferentes porque son **diferentes objetos**. Para que en el segundo caso muestre la cadena "SON IGUALES" debería de cambiarse la línea de código:

```
String cad2=new String("EMMA");
```

Por la siguiente otra:

```
String cad2=cad1;
```

En este último caso es importante recalcar que *cad2* y *cad1* apuntan al mismo objeto.

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

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

El código anterior mostrará por pantalla la cadena "MAYKA".

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

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

El código anterior mostrará por pantalla la cadena "pedro ruiz".

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

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

El código anterior mostrará por pantalla la cadena “JUAN SERRANO”.

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

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

El código anterior mostrará por pantalla la cadena “JOAN SOAREZ”.

- **String substring(int i, int f)**. Este método devuelve un nuevo objeto *String* que será la subcadena que comienza en el carácter *i* y termina en el carácter *f* (el carácter *f* 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));
```

El código anterior mostrará por pantalla las cadenas “CARLOS” y “MORENO”.

- **boolean startsWith(String cad)**. Este 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"));
```

El código anterior mostrará por pantalla *false* y *true*.

- **boolean endsWith(String cad)**. Este 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"));
```

El código anterior mostrará por pantalla *false*, *true* y *false*. La primera vez muestra *false* porque aunque ‘p’ y ‘P’ son la misma letra, Java las trata de manera diferente al ser dos símbolos ASCII distintos.

- **char charAt(int pos)**. Devuelve el carácter del objeto *String* que se especifica en el parámetro *pos*.

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

El código anterior mostrará por pantalla la cadena “A H”.

**iCuidado!**

Si en la función `charAt` se utiliza un índice que no está entre los valores 0 y `length()-1`, Java lanzará una excepción.

- **int indexOf(int c) o int indexOf(String s)**. Este método admite dos tipos de parámetros y nos permite encontrar la primera ocurrencia de un carácter o una subcadena dentro de un objeto del tipo `String`. En el caso de que no sea encontrado el carácter o la subcadena este método devolverá 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"));
```

El código anterior mostrara por pantalla el siguiente resultado: 1, -1, 5 y -1.

- **char[] toCharArray()**. Este método devuelve un vector o array de caracteres a partir del propio objeto `String`.

```
String cad1 = "LORO FELIPE";
char cad2 []=cad1.toCharArray();
```

El código anterior creará un array de caracteres `cad2` que contendrá la cadena “LORO FELIPE” contenida en el objeto `String` `cad1`.

- **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 edad3=6;
str=String.valueOf(edad3);
double edad4=6.5;
str=String.valueOf(edad4);
```

A FONDO

CONVERTIR UN STRING EN UN NÚMERO

```
String snumero=" 6  ";
int numero=Integer.parseInt(snumero.trim());
//int numero=Integer.parseInt(snumero);
```

Con el código anterior es posible convertir un objeto String en un número entero. La tercera línea de código está comentada porque lanzaría una excepción dado que no se han limpiado los espacios a derecha e izquierda de la cadena y contiene caracteres no numéricos.

Si el número es un decimal y no se quieren perder los decimales se utilizaría el siguiente código:

```
String snumero=" 6.5  ";
double numero=Double.valueOf(snumero).doubleValue();
```

7.3.2 LA CLASE STRINGBUFFER

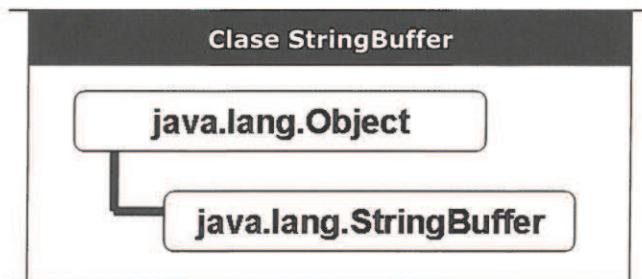


Figura 7.4. Jerarquía de la clase `StringBuffer`



Recuerda

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 apellidos = new StringBuffer(80);
StringBuffer direccion = new StringBuffer();
```

La segunda línea de código crea un objeto vacío con una capacidad para 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 *StringBuffer*.

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

Curioso: La salida por pantalla del código anterior será 20 aunque su longitud era solo de 4. Eso es debido a que cuando se crea un 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 *StringBufffer*. El tipo del argumento puede ser *int, long, float, double, boolean, char, char[], String* y *Object*.

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

El código anterior muestra por pantalla la cadena “Juan Carlos Moreno Pérez”.

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

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

El código anterior muestra por pantalla la cadena “EMMA MORENO”.

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

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

El código anterior mostrara por pantalla la cadena “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 = nombre.delete(4,10);
System.out.println(nombre);
```

El código anterior mostrará por pantalla la cadena de caracteres “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 = nombre.replace(5,10,"TURRION");
System.out.println(nombre);
```

El código anterior mostrará por pantalla la cadena de caracteres “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 si no se especifica el argumento *y*).

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

El código anterior mostrará por pantalla la cadena de caracteres “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 por pantalla.

- **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 mostrará 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 anterior mostrará por pantalla la cadena de caracteres “eMMA”.

A FONDO

CLASE STRINGTOKENIZER

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

Un ejemplo de utilización de esta clase es el siguiente:

```
 StringTokenizer str;
 str = new StringTokenizer("UNO DOS TRES PERICO JUANICO Y_ANDRES");
 System.out.println("La cadena str tiene "+str.countTokens()+" elementos");
 while (str.hasMoreTokens()) System.out.println(str.nextToken());
```

El código anterior mostrará que la cadena str tiene 6 elementos y los irá sacando por pantalla uno a uno.

Para utilizar esta clase se debe de importar la clase *StringTokenizer*:

```
import java.util.StringTokenizer;
```

O importar todas las clases del paquete *java.util*:

```
import java.util.*;
```

También es posible especificar los delimitadores dentro del constructor de la clase:

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

En el ejemplo anterior se utiliza el símbolo '|' como delimitador.

7.4 ARRAYS O VECTORES DE OBJETOS STRING

Dado que ya se conoce cómo funcionan los vectores y los objetos *String*, en este apartado se va a ver un ejemplo de utilización de estos tipos de datos. En el ejemplo se va a realizar un programa que lee nombres por teclado y los va almacenando en un vector. Una vez creado el vector se mostrarán dichos nombres por teclado según la posición en que se han leído.

```
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.print(lista[i] + " ");
    }
public static void main(String[] args) {
    lista = new String[POS];
    for (int i=0;i<POS;i++){
        String ln = System.console().readLine();
        lista[i]=ln.toString();
    }
    System.out.println("");
    muestra();
    System.out.println("");
}
}

```

7.5 ALGORITMOS DE ORDENACIÓN

Los algoritmos de ordenación se aplican generalmente en arrays unidimensionales (también en ficheros) y su finalidad es organizar los datos en dichos arrays. Estos algoritmos de ordenación tienen gran importancia, con lo cual en muchos lenguajes de programación existen librerías que contienen funciones o procedimientos para ordenar arrays. No todos los algoritmos ordenan de la misma forma ni todos son igual de eficientes, algunos son muy rápidos cuando el vector está casi ordenado, otros lo son cuando está muy desordenado y a otros el preordenamiento no les afecta en gran medida. Dependiendo de la estrategia, algunos algoritmos son mejores que otros. Los algoritmos se distinguen por su complejidad computacional la cual clasifica los algoritmos dependiendo de su eficiencia, en esta clasificación se tiene en cuenta el peor caso, el caso promedio y el mejor de los casos.

Tabla 7.2. Algoritmos de ordenación

Algoritmo	Estrategia de ordenación
Burbuja o bubblesort	Este ordenamiento pega una serie de pasadas al vector y compara parejas de elementos adyacentes y los intercambia si no están ordenados. En la primera pasada obviamente el mayor elemento se situará el último y así sucesivamente. Cuando en una pasada no hace ningún intercambio quiere decir que el vector ya está ordenado.
Cocktail sort	Es un ordenamiento por burbuja bidireccional.

Ordenación por inserción o insertion sort	La ordenación se realiza insertando los datos ordenadamente en un vector. Los datos se van insertando agrupados y cuando se inserta un nuevo dato se le hace hueco en la posición que le corresponde desplazando los demás elementos.
Por Mezcla o Merge Sort	Es un algoritmo desarrollado por Von Newmann que emplea la técnica divide y vencerás. El algoritmo va dividiendo por la mitad de forma recursiva el vector a ordenar en dos listas las cuales deberán de ser también ordenadas. El algoritmo para de dividir cuando se queda con 0 ó 1 elementos, entonces se consideran que están ya ordenados. Si se ordenan dos listas las cuales a su vez están ordenadas, el resultado será una lista ordenada.
Ordenamiento por selección o selection sort	El funcionamiento es simple, se busca el menor elemento de la lista y se coloca en el primer lugar, luego se busca el segundo y se coloca en el segundo lugar y así sucesivamente.
Ordenación rápida o Quicksort	Es el algoritmo más rápido. Utiliza la técnica divide y vencerás. Es un algoritmo recursivo que utiliza un pivote como elementos central y coloca todos los elementos menores a su izquierda y los mayores a su derecha. La lista se separa en dos sublistas y se repite el proceso de manera recursiva con las dos sublistas hasta que toda la lista se queda ordenada.

7.5.1 ORDENACIÓN POR EL MÉTODO DE LA BURBUJA

El ordenamiento por el método de la burbuja es muy utilizado, sobre todo porque es sencillo de entender e implementar. En la siguiente tabla se puede comprender claramente cómo funciona el método de la burbuja. Como se puede apreciar, al igual que las burbujas de aire en el agua suben a la superficie, los elementos mayores de la lista se irán colocando al final del vector de forma ordenada.

1^a Pasada

4	2	9	3	1
2	4	9	3	1
2	4	3	9	1
2	4	3	1	9

2^a Pasada

2	4	3	1	9
2	3	4	1	9
2	3	1	4	9

3^a Pasada

2	3	1	4	9
2	1	3	4	9

4^a Pasada

2	1	3	4	9
1	2	3	4	9

Como parece lógico, las pasadas cada vez serán más cortas dado que en cada pasada, al menos 1 elemento quedará ordenado al final de la lista. Una implementación de este código en Java sería el siguiente:

```
public static void burbuja(int array[]){
    int aux;
    for (int i=array.length;i>0;i--){
        for (int j=0;j<i-1;j++){
            if (array[j]>array[j+1]){
                aux = array[j+1];
                array[j+1]=array[j];
                array[j]=aux;
            }
        }
    }
}
```

Este código no es del todo eficiente porque si el vector ya está ordenado el procedimiento seguirá comprobando hasta terminar sin tener en cuenta este hecho. Para conseguir esto, basta con utilizar un *flag* que marque si el vector está ordenado o no y en el caso de que el algoritmo de una pasada sin realizar ningún intercambio de elementos el proceso parará.

A FONDO

BÚSQUEDA DE DATOS DENTRO DE UN ARRAY

Las ventajas de los arrays o vectores son la versatilidad y posibilidades que ofrecen. Una de las operaciones más frecuentes que se realizan en los mismos 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 la más eficiente. Imaginemos un array muy grande. De media tendremos que buscar en $N/2$ elementos para encontrar nuestro dato. Si el array es muy grande perderemos mucho tiempo durante la búsqueda. Un método más eficiente es la **búsqueda binaria**. En la búsqueda binaria se sigue el lema "divide y vencerás". En la siguiente imagen se puede observar cómo funciona el algoritmo:

Elemento a buscar: 5

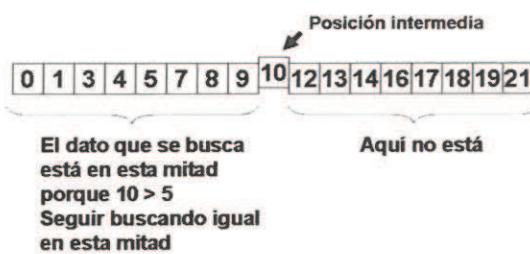


Figura 7.5. Búsqueda binaria

Importante: Los datos del array donde se va a realizar la búsqueda deberán de estar ordenados.

Como se puede ver, lo que se hace es elegir una posición intermedia y desechar de la búsqueda la mitad del array donde el dato no va a estar siguiendo la búsqueda en el lado restante.

La búsqueda seguirá igual pero para la mitad del array donde debería de estar el elemento a buscar.

7.5.2 ORDENACIÓN POR EL MÉTODO DE INSERCIÓN DIRECTA

Este algoritmo funciona de la siguiente manera, se desean insertar en un array de 5 elementos los siguientes datos (4, 2, 9, 3 y 1).

1^a Inserción (4)

4				
---	--	--	--	--

2^a Inserción (2)

4				
	4			
2	4			

3^a Inserción (9)

2	4			
2	4	9		

4^a Inserción (3)

2	4	9		
2		4	9	
2	3	4	9	

5^a Inserción (1)

2	3	4	9	
	2	3	4	9
1	2	3	4	9

A FONDO

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.

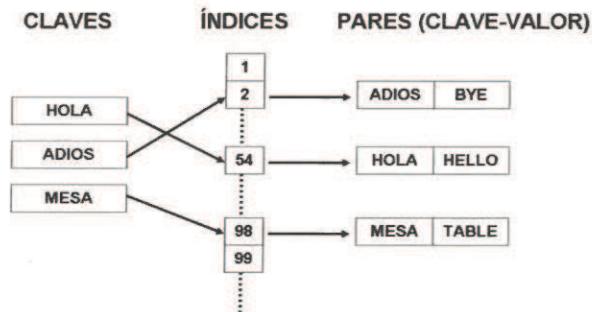


Figura 7.6. La clase Hashtable

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

```

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 por pantalla
        System.out.println("ADIOS : " + despedida); //muestra BYE por pantalla
        System.out.println("BRAZO : " + brazo); //muestra null por 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 datos de la tabla hash... ");
        Enumeration k = dic.keys();
        while( k.hasMoreElements() ) System.out.println( k.nextElement() );
        System.out.println("Mostrando todos los elementos de la tabla hash... ");
    }
}
  
```

```
Enumeration e = dic.elements();
while( e.hasMoreElements() ) System.out.println( e.nextElement() );
System.out.println( "Eliminando el dato " + dic.remove("HOLA"));
}
}
```



RESUMEN DEL CAPÍTULO



En este tema se estudia el concepto de vector o array. Existen numerosos problemas que se resuelven con estructuras de este tipo, por lo tanto es de obligado cumplimiento estudiarlo. También se estudia en profundidad las cadenas de caracteres que ya se vieron en capítulos anteriores. Estudiar arrays y no ver algoritmos de ordenación es un pecado, por lo tanto, se estudiarán dos de los más útiles como son el método de la burbuja, el cual es sencillo de comprender y el método de inserción directa. Existen multitud de algoritmos mucho más eficientes y rápidos que el de la burbuja. Se aconseja al alumno que busque alguno por Internet e intente comprenderlo.



EJERCICIOS RESUELTOS



- 1. Realiza un programa que genere una matriz 5 x 8 y muestre los elementos en forma de matriz.

```
public class matriz {
    public static void main(String[] args) {
        int[][] matriz = new int[5][8];
        for (int i=0;i<5;i++) {
            for (int j=0;j<8;j++) {
                matriz[i][j]=i+j;
            }
        }
        for (int i=0;i<5;i++) {
            for (int j=0;j<8;j++) {
                System.out.print(matriz[i][j]);
            }
        }
    }
}
```