

U.D.5: TABLAS

INTRODUCCIÓN

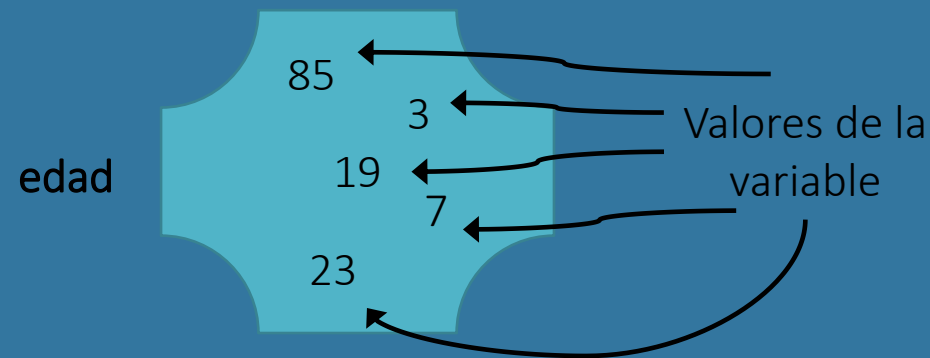
Una ***variable solo puede almacenar un valor simultáneamente*** en cada instante de tiempo. A este tipo de variables ***se les denomina variables escalares.***

Existe una forma de ***almacenar más de un valor simultáneamente en una variable y es mediante el uso de tablas.***

A las tablas se les suele denominar con los nombres: arrays (en inglés) o vectores.

VARIABLES ESCALARES VERSUS TABLAS

Una tabla es una variable que permite guardar más de un valor simultáneamente. Podríamos ver una *tabla* como una “supervariable” que engloba a otras variables, llamadas elementos o componentes referidas con un mismo nombre, con la **condición** de que **todas sean del mismo tipo**.



VARIABLES ESCALARES VERSUS TABLAS

Siempre que necesitemos manejar varios datos del mismo tipo simultáneamente, en general, se recomienda utilizar una tabla en lugar de varias variables escalares.

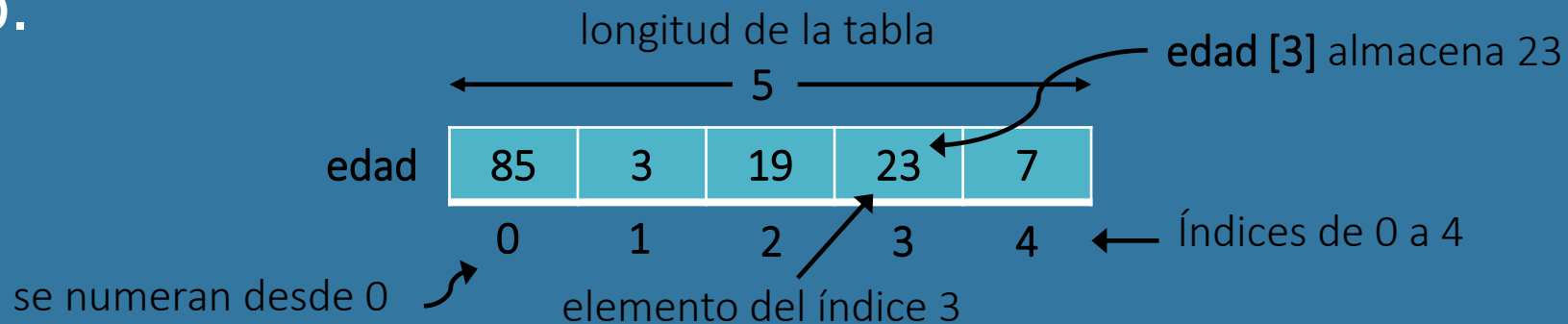
Es mucho más cómodo trabajar con una tabla que almacena, por ejemplo, 100 valores, que hacerlo con 100 variables escalares.

Cuando desconocemos cuántos datos son necesarios gestionar, no existe otra alternativa que utilizar tablas, ya que, a la hora de crearlas, el número de elementos que guardaremos en ella se puede elegir dinámicamente.

ÍNDICES

La forma de distinguir entre cada uno de los elementos o componentes que constituyen una tabla, es mediante la asignación de un número de orden a cada elemento.

A este número se le llama índice. Al primer elemento se le asigna el índice 0, al segundo el índice 1 y así sucesivamente. Al último elemento le corresponde como índice el número total de elementos menos uno.



ÍNDICES

La forma de *utilizar* un *elemento concreto de una tabla* es *por medio del nombre de la variable* que identifica a la tabla *junto al número, “índice”, entre corchetes ([])* que distingue ese elemento.

Por ejemplo, para utilizar el cuarto elemento de la tabla edad, elemento con índice 3, escribiremos edad[3], que contiene un valor de 23.

ÍNDICES

Veamos un ejemplo de cómo mostrar y asignar un elemento:

```
System.out.println(edad[0]); // muestra el primer elemento: 85  
edad[3] = 8; // asigna un nuevo valor al cuarto elemento.
```

ÍNDICES FUERA DE RANGO

La variable edad es una tabla de 5 enteros y podemos utilizar cada uno de los cinco elementos que la componen de la forma: edad[0], edad[1], ... , edad[4].

¿Qué ocurrirá si utilizamos un índice que se encuentra fuera del rango de 0 a 4 ?. Es decir, ¿qué efectos producen edad[-2] o edad[7]?. En ambos casos obtendremos un error en tiempo de ejecución que provoca que el programa termine de forma inesperada, ya que detecta que los elementos con los índices utilizados no existen.

ÍNDICES FUERA DE RANGO

La mayoría de los errores al trabajar con tablas provienen de utilizar índices fuera de rango. Hay que prestar especial atención a esto.

CONSTRUCCIÓN DE TABLAS

En el momento de crear una tabla, debemos tener en cuenta lo siguiente:

- Decidir qué tipo de datos vamos a almacenar y cuántos elementos necesitamos.
- Declarar una variable para la tabla.
- Crear la propia tabla.

LONGITUD Y TIPO

Una tabla se define mediante dos características fundamentales: su longitud y su tipo.

La longitud es el número de elementos que tiene, y el tipo de una tabla es el de los datos que almacena en todos y cada uno de sus elementos.

En la siguiente figura podemos ver dos tablas: la primera compuesta por 3 elementos de tipo double y la segunda por seis elementos de tipo int.

LONGITUD Y TIPO

1.27	0.3	9.0
0	1	2

longitud 3 tipo double

5	-1	0	2	0	-7
0	1	2	3	4	5

longitud 6 tipo int

La primera tabla podría utilizarse para almacenar, por ejemplo, el tiempo empleado en realizar algunas pruebas, mientras que la segunda tabla podría usarse para guardar el número de puntos obtenidos en distintas partidas de un videojuego.

VARIABLES DE TABLA

El primer paso para crear una tabla es declarar la variable utilizando corchetes ([]), símbolo que diferencia entre una variable escalar y una que es tabla. Es posible utilizar dos sintaxis equivalentes:

tipo nombreVariable[];

tipo [] nombreVariable;

VARIABLES DE TABLA

Donde tipo representa cualquier tipo primitivo. Veamos cómo declarar la variable edad como una tabla de tipo int:

```
int edad[ ];
```

o mediante la forma (ambas son equivalentes):

```
int [ ] edad;
```

En este punto la variable está declarada, pero no hemos construido ninguna tabla.

OPERADOR new

Una vez que hemos declarado una variable, crearemos una tabla con una longitud adecuada y la asignaremos a la variable. La sintaxis es:

nombreVariable = new tipo[longitud];

Veamos cómo crear la tabla edad, de tipo int con una longitud de 5 elementos:

Edad = new int[5];

El operador new construye una tabla donde todos los elementos se inicializan a 0, para tipos numéricos, o false si la tabla es booleana.

OPERADOR new

Nota: Los elementos de las tablas de otros tipos se inicializan a null.

Es posible declarar la variable y crear la tabla en una única sentencia.

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

Existe una alternativa para crear una tabla sin necesidad de utilizar el operador new. En la misma declaración se asignan valores a los elementos de la tabla, que se crea con la longitud necesaria para albergar todos los valores asignados. Por ejemplo:

```
int datos[ ] = {2, -3, 0, 7}; // tabla de longitud 4
```

OPERADOR new

Esta sentencia declara la variable `datos` y crea una tabla de cuatro enteros, donde cada elemento tiene asignado el valor correspondiente, equivalente a:

```
int datos[ ]; // declaramos la variable
```

```
datos = new int[4]; // creamos la tabla
```

```
datos[0] = 2; // asignamos valores
```

```
datos[1] = -3;
```

```
datos[2] = 0;
```

```
datos[3] = 7;
```

OPERADOR *new*

La creación de una tabla mediante la asignación de valores solo puede realizarse en la misma instrucción donde se declara. No se puede escribir.

```
int datos[ ];  
datos = {2, -3, 0, 7}; // ¡Error! Solo en la declaración
```

REFERENCIAS

La siguiente instrucción:

```
edad = new int [10];
```

construye una tabla de 10 elementos de tipo int y la asigna a la variable edad. Estudiemos cómo funciona el operador new:

1. En primer lugar, calcula el tamaño físico de la tabla, es decir, el número de bytes que ocupará la tabla en la memoria. Este calculo es sencillo y se obtiene de multiplicar la longitud de la tabla por el tamaño de su tipo. Como ejemplo vamos a calcular el tamaño físico de una tabla de longitud diez de tipo int:

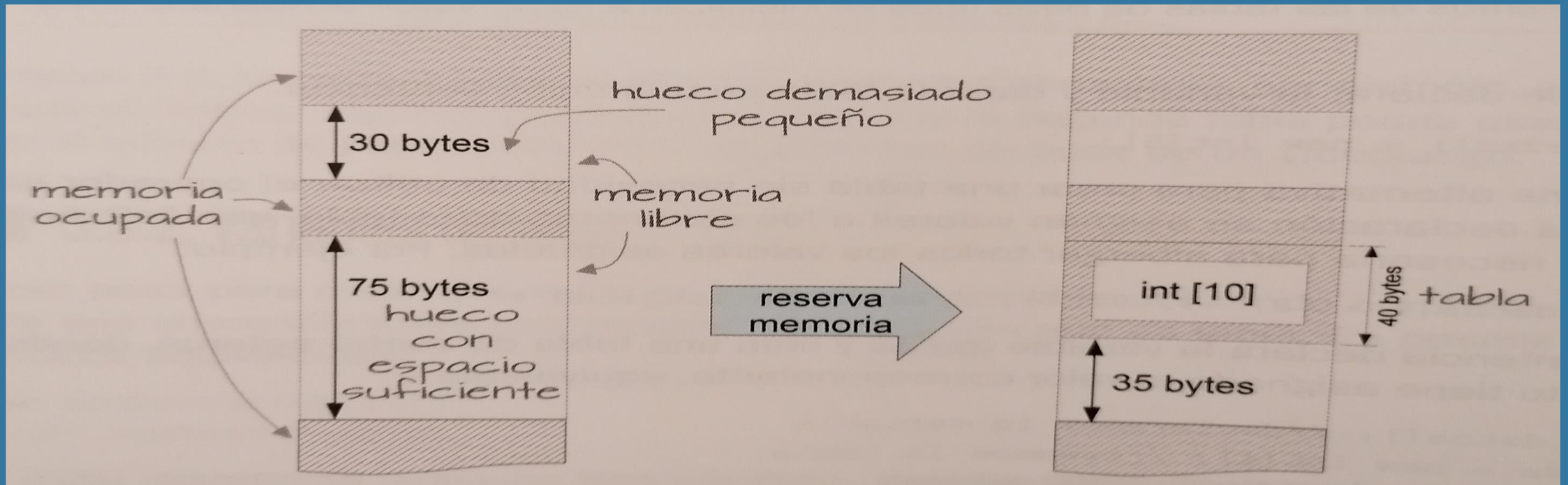
REFERENCIAS

**tamaño en memoria = n°. elementos tabla * tamaño tipo (int)
= 10 * 4 bytes = 40 bytes**

2. Conociendo el tamaño físico de la tabla, busca en la memoria un hueco libre (memoria no utilizada), con un tamaño suficiente para albergar todos los elementos de la tabla consecutivamente.
3. Reserva la memoria necesaria para almacenar la tabla y la marca como memoria ocupada, siendo este el sitio donde se almacenarán los elementos de la tabla.

REFERENCIAS

4. Por último, recorre todos los elementos de la tabla inicializándolos de la siguiente manera: *0 si es una tabla numérica, false si la tabla es booleana.*



REFERENCIAS

En este punto hemos creado la tabla. El siguiente paso es asignarla a la variable correspondiente; para ello Java dispone de un mecanismo para indicar dónde está la tabla en la memoria.

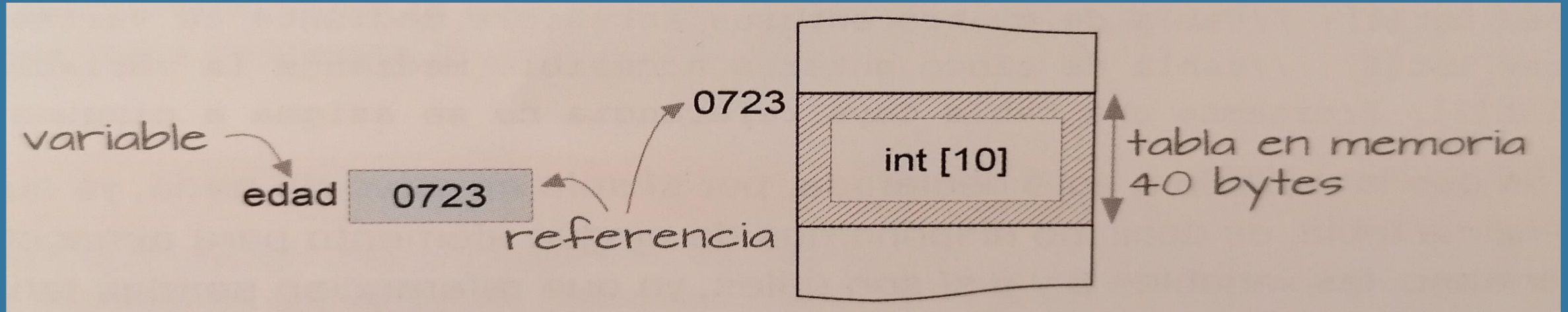
Cada posición de la memoria de un ordenador tiene una dirección única que la identifica. Así, la primera posición de memoria tiene la dirección 000; la siguiente, la dirección 001, y así sucesivamente. En Java a cada dirección de memoria se le denomina ***referencia***.

REFERENCIAS

La forma de que una variable sepa dónde está la tabla en la memoria es asignándole la referencia de la primera posición que ocupa (una tabla puede ocupar varias posiciones consecutivas). Lo que almacenan realmente las variables de tabla son referencias.

Por este motivo se les conoce también como variables de referencia. En la siguiente figura se muestra la zona de la memoria reservada para la nueva tabla, que empieza en la dirección, por ejemplo, 0723. Por tanto, esa es la referencia de la tabla.

REFERENCIAS



Si ejecutamos las siguientes líneas:

```
int t[ ] = new int[10];  
System.out.println(t);
```

REFERENCIAS

Podríamos pensar (erróneamente) que se muestra por consola el valor de cada elemento de la tabla `t`, pero esto no es así. Lo que se muestra es la referencia que guarda la variable `t`, que suele tener una forma similar a: `l@659e0bfd`.

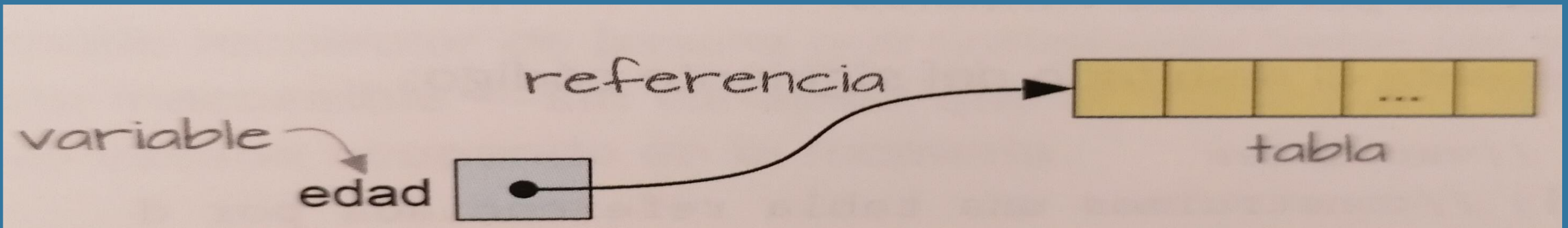
Curiosidades

La `<<l>>` de la referencia especifica que la tabla es de enteros (`int`). El primer carácter identifica el tipo de la tabla: `B` para `byte`, `D` para `double`, `Z` para `booleanos`, etcétera.

A partir de `@` se muestra la dirección de memoria (en hexadecimal) en la que se encuentra la tabla. En nuestro ejemplo: `659e0bfd`.

REFERENCIAS

Las referencias se modifican en cada ejecución, dependiendo de la ocupación de la memoria. En las representaciones gráficas es mucho más intuitivo sustituir los valores de la referencia por una flecha, que tiene el mismo significado: “la variable está referenciando esta tabla”. Esta representación se muestra en la siguiente figura, donde hemos cambiado el bloque de memoria por casillas que representan los elementos de la tabla.



REFERENCIAS

Ahora que entendemos cómo funcionan las referencias, podemos analizar el siguiente código:

```
int a[ ], b[ ], c[ ]; // variables
```

```
b = new int [4]; // tabla de 4 enteros accesible mediante la variable b
```

```
c = new int [5]; // tabla de 5 enteros accesible mediante la variable c
```

```
new int [3]; // creamos una tabla cuya referencia no se asigna a ninguna variable.
```

REFERENCIAS

A pesar de que la variable a está declarada, por sí sola no sirve de nada, ya que no referencia ninguna tabla, es decir, no disponemos de ningún elemento para almacenar datos.

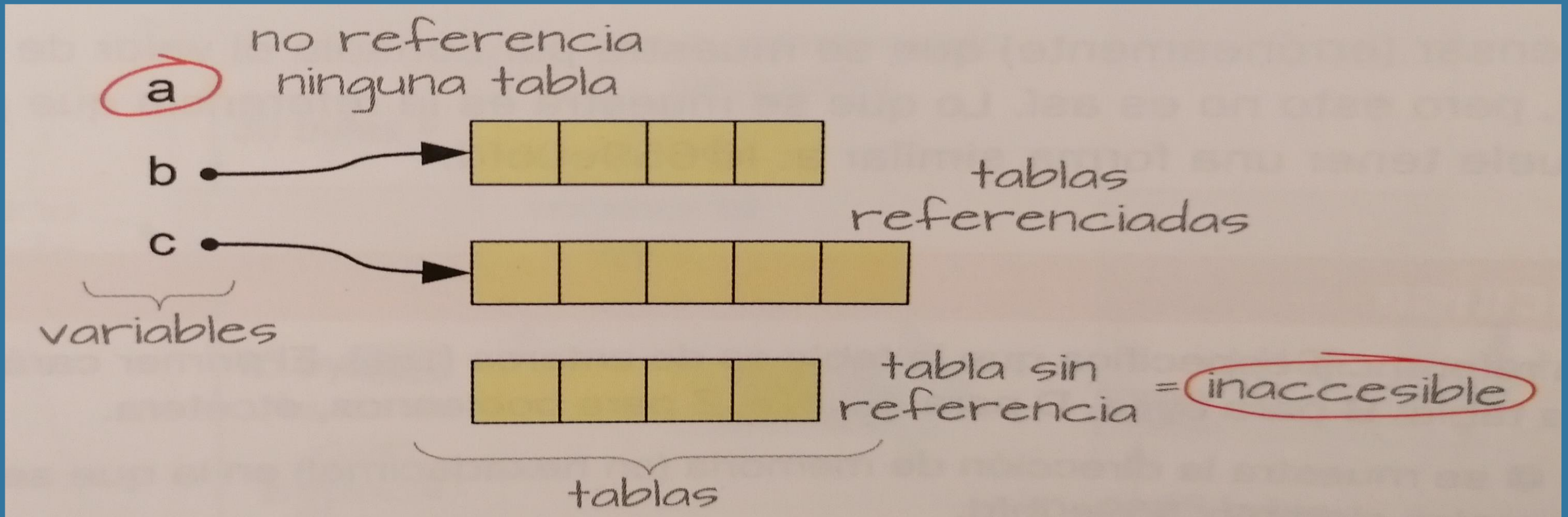
Por el contrario, las variables b y c sí son útiles, ya que refieren sendas tablas.

REFERENCIAS

La última instrucción (**new int[3];**) construye una tabla con tres elementos enteros, pero la referencia que devuelve new no se asigna a ninguna variable, lo que convierte la tabla en inútil, ya que es inaccesible. Una vez que hemos perdido la referencia de una tabla, no existe forma alguna de recuperarla.

REFERENCIAS

La siguiente figura muestra todos los casos posibles de referencias.



REFERENCIAS

Las variables pueden verse como medios para acceder a las tablas a las que referencian. Es posible acceder a una misma tabla mediante más de una variable; para ello, la tabla debe estar referenciada por estas variables.

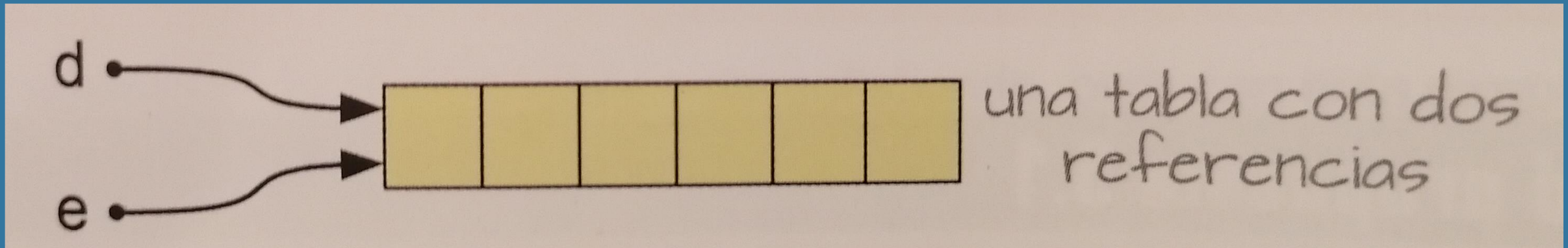
```
int d[ ], e[ ]; // variables
```

```
d = new int [6]; // construimos una tabla referenciada por d.
```

```
e = d; // ahora la variable e referencia la misma tabla que d.  
      // Ambas guardan la misma dirección de memoria.
```

REFERENCIAS

En la siguiente figura se representa el resultado del código anterior.



REFERENCIAS

Ahora podemos acceder a los mismos datos utilizando la variable `d` o la variable `e`: utilizar `d[2]` es equivalente a utilizar `e[2]`, ya que `d` y `e` referencian la misma tabla. De todas formas esta práctica es poco aconsejable, ya que puede producir cambios no deseados en la tabla. La única condición para que una variable pueda referenciar a una tabla es que el tipo de ambas coincidan.

```
int d[ ], e[ ]; // variables
```

```
d = new int [6]; // construimos una tabla referenciada por d.
```

```
e = d; // ahora la variable e referencia la misma tabla que d.  
      // Ambas guardan la misma dirección de memoria.
```

REFERENCIAS

El siguiente código es erróneo debido a que los tipos no coinciden:

```
boolean t1[ ]; // variable para tablas booleanas
```

```
int t2[ ]; // variable para tablas enteras
```

```
t1 = new boolean[10]; // construye y asigna una tabla de 10 booleanos
```

```
t2 = t1; // ¡ERROR! Tipos incompatibles
```

```
// t2 puede referenciar tablas enteras, pero no booleanas.
```

RECOLECTOR DE BASURA

¿Qué ocurre cuando una tabla no está referenciada por ninguna variable?. Lo primero y más obvio es que dicha tabla es inútil; no hay forma de acceder a sus elementos. Pero existe un segundo problema: la tabla está ocupando espacio en la memoria.

Quizá el tamaño de unas pocas tablas inútiles en la memoria no sea significativo, pero una aplicación puede dejar, durante su ejecución, grandes cantidades de memoria ocupada inaccesible. Esto puede ocurrir por un mal diseño o de forma malintencionada.

RECOLECTOR DE BASURA

Java soluciona el problema mediante un mecanismo muy ingenioso: periódicamente se inicia un proceso llamado recolector de basura que comprueba todas las tablas construidas. Si encuentra alguna inaccesible (sin variables que la referencie) la destruye, dejando libre el espacio que estaba ocupando en la memoria.

RECOLECTOR DE BASURA

Java soluciona el problema mediante un mecanismo muy ingenioso: periódicamente se inicia un proceso llamado recolector de basura que comprueba todas las tablas construidas. Si encuentra alguna inaccesible (sin variables que la referencie) la destruye, dejando libre el espacio que estaba ocupando en la memoria.

REFERENCIA NULL

Hemos visto la forma de asignar a una variable la referencia de una tabla: al crearla con el operador `new` o a través de otra variable. Pero existe la forma de hacer justo lo contrario, es decir, hacer que una variable que referencia a una tabla no referencie nada. Para ello disponemos del literal *null*, que significa *vacío*.

REFERENCIA NULL

Veamos un ejemplo de cómo dejar sin referencia a una tabla.

```
int t1[ ], t2[ ]; // variables de tipo tabla entera
```

```
t1 = new int [100]; // t1 referencia una tabla de 100 elementos.
```

```
t2 = t1; // ahora t2 también referencia la misma tabla.
```

```
t1 = null; // anulamos t1: no referencia nada.
```

```
    // la tabla sigue siendo accesible desde t2.
```

```
t2 = null; // anulamos t2: tampoco hace referencia a nada.
```

```
// la tabla es inaccesible: el recolector de basura se encarga de ella.
```

REFERENCIA NULL

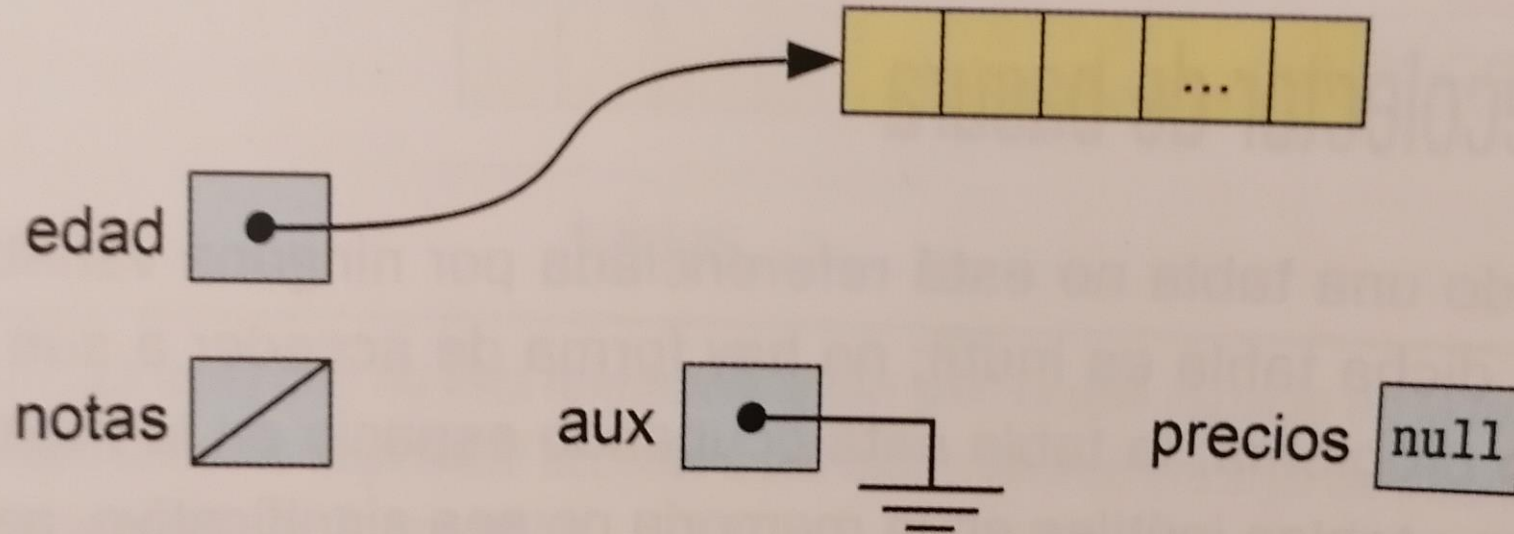


Figura Mientras una referencia se suele representar con una flecha entre la variable y la tabla, es habitual encontrar la referencia vacía representada mediante una línea cruzada, una toma de tierra o incluso con la propia palabra `null`. Todo ello informa que la variable no está referenciando nada.

USO DE TABLAS

Existen distintas técnicas para trabajar con tablas:

- Podemos considerar que solo algunos elementos de una tabla almacenan información útil mientras la información de otros elementos no es relevante.
- Otra alternativa es suponer que todos los elementos de una tabla contienen siempre información útil y es la tabla la que adapta su longitud a las necesidades del momento.

USO DE TABLAS

Esta segunda técnica simplifica la resolución de los problemas y permite aprovechar las herramientas que proporciona Java, lo que minimiza las implementaciones propias que tenemos que desarrollar.

USO DE TABLAS

Veamos un ejemplo donde todos los elementos de una tabla contienen siempre datos útiles, sea la variable estatura que referencia una tabla con la altura de algunas personas:

estatura	1,23	2,07	1,74	1,86	1,35
	0	1	2	3	4

USO DE TABLAS

Las tablas, una vez creadas, mantienen su longitud constante y no es posible cambiar el número de elementos que contienen. Si necesitamos modificar la longitud de una tabla, lo que haremos será crear una segunda tabla con el número de elementos necesarios y copiar en ella los datos que nos interesan de la tabla original.

USO DE TABLAS

Si la nueva tabla se referencia con la misma variable que referenciaba a la original, a efectos prácticos es como si la tabla original hubiera modificado su longitud.

Por tanto, si deseamos modificar la longitud de **estatura**, tendremos que crear una segunda tabla que estará referenciada por la misma variable **estatura**, dando la sensación de que la longitud de la tabla ha cambiado.

Indistintamente de la opción elegida, podemos optar por mantener cierto orden entre los datos de una tabla.

TABLAS ORDENADAS

En ocasiones, interesa que los datos estén ordenados siguiendo algún criterio. Un ejemplo de una tabla ordenada en sentido creciente es:

precios	12,3	18,0	34,65	80,19	102,0
	0	1	2	3	4

Cuando usamos tablas ordenadas, es muy importante, en el momento de insertar o eliminar un elemento, realizar la operación teniendo cuidado de que la tabla continúe ordenada.

TABLAS + INDICADOR

Otra técnica para usar las tablas es suponer que no todos sus elementos almacenan datos. La tabla estará subdividida en dos partes: la primera con los elementos que almacenan datos y la segunda con los elementos vacíos.

Para ello, la tabla irá acompañada de una variable entera que funciona como indicador del número de datos que contiene la tabla, es decir, el indicador especifica cuántos elementos forman la primera parte (datos útiles) y el resto se consideran elementos vacíos.

TABLAS + INDICADOR

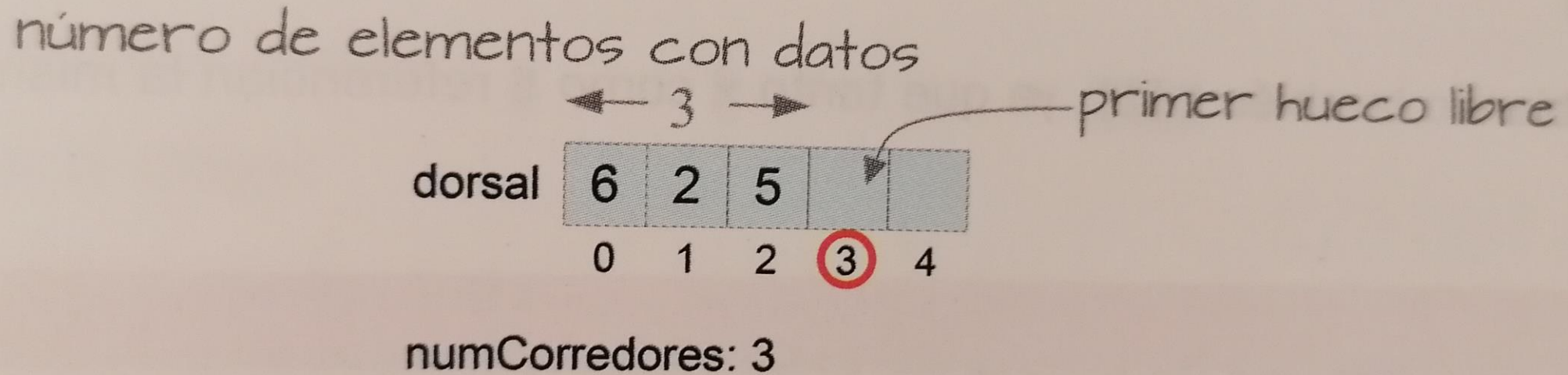


Figura Tabla de longitud 5 donde solo se almacenan 3 dorsales (solo tres datos). La variable `numCorredores` actúa como indicador de la tabla. En caso de insertar un nuevo corredor se usaría el primer hueco (elemento) libre (con índice 3).

TABLAS + INDICADOR

Con esta técnica, cada vez que se inserta o elimina un dato en la tabla no es necesario modificar su longitud; en su lugar se modifica el indicador, lo que hace que el número de elementos con datos aumente o disminuya.

Los elementos de una tabla siempre almacenan algún valor, por lo que los elementos que se consideran vacíos realmente contienen valores (denominados valores basura) que nunca tendremos en cuenta.

TABLAS + INDICADOR

Aunque para esta unidad optaremos por usar tablas en las que todos los elementos contienen datos útiles, se ha comentado la técnica de **tabla + indicador** porque también la podéis encontrar si consultáis bibliografía. *Su principal ventaja es que no necesita redimensionar continuamente la tabla, pero en cambio, debemos llevar un control manual de indicador.*

TABLAS COMO PARÁMETROS DE FUNCIONES

Si recordamos cuál es el mecanismo de paso de parámetros al invocar una función: el valor de la variable que se utiliza en la llamada se copia al parámetro de la función (que es una variable local en la función). Veamos un ejemplo:

```
muestra (a); // llamada a la función
```

```
...
```

```
void muestra (int b) { // definición de la función
```

```
...
```

```
}
```

TABLAS COMO PARÁMETROS DE FUNCIONES

El valor de la variable *a* se copia en el parámetro *b*. Si en el cuerpo de la función se modifica la variable *b*, ***se está modificando una copia, no la variable a.***

Cuando utilizamos tablas como parámetros el mecanismo es el mismo, es decir, el valor de la variable utilizada en la llamada se copia al parámetro de la función.

Pero en este caso, lo que se copia es la referencia de una tabla, con lo que se consigue que la tabla esté referenciada tanto por la variable utilizada en la llamada como por el parámetro.

TABLAS COMO PARÁMETROS DE FUNCIONES

La modificación de un elemento de la tabla dentro de la función es visible desde la referenciada externa a la función. Esto es normal, ya que disponemos de dos referencias, pero de una única tabla donde se realizan las modificaciones.

En el ejemplo de la siguiente figura, si dentro del cuerpo de la función `ejemploFuncion()` ejecutamos:

`x[2] = 10;`

Estamos cambiando también **`t[2]`**, ya que tanto **`x`** como **`t`** referencian la misma tabla.

TABLAS COMO PARÁMETROS DE FUNCIONES

Recordatorio: Las funciones solo pueden devolver un único valor mediante la instrucción return. El uso de tablas como parámetros y el hecho que se compartan sus datos entre la función y quien la invoca, permite que una función pueda devolver más información que con un simple return.

TABLAS COMO PARÁMETROS DE FUNCIONES

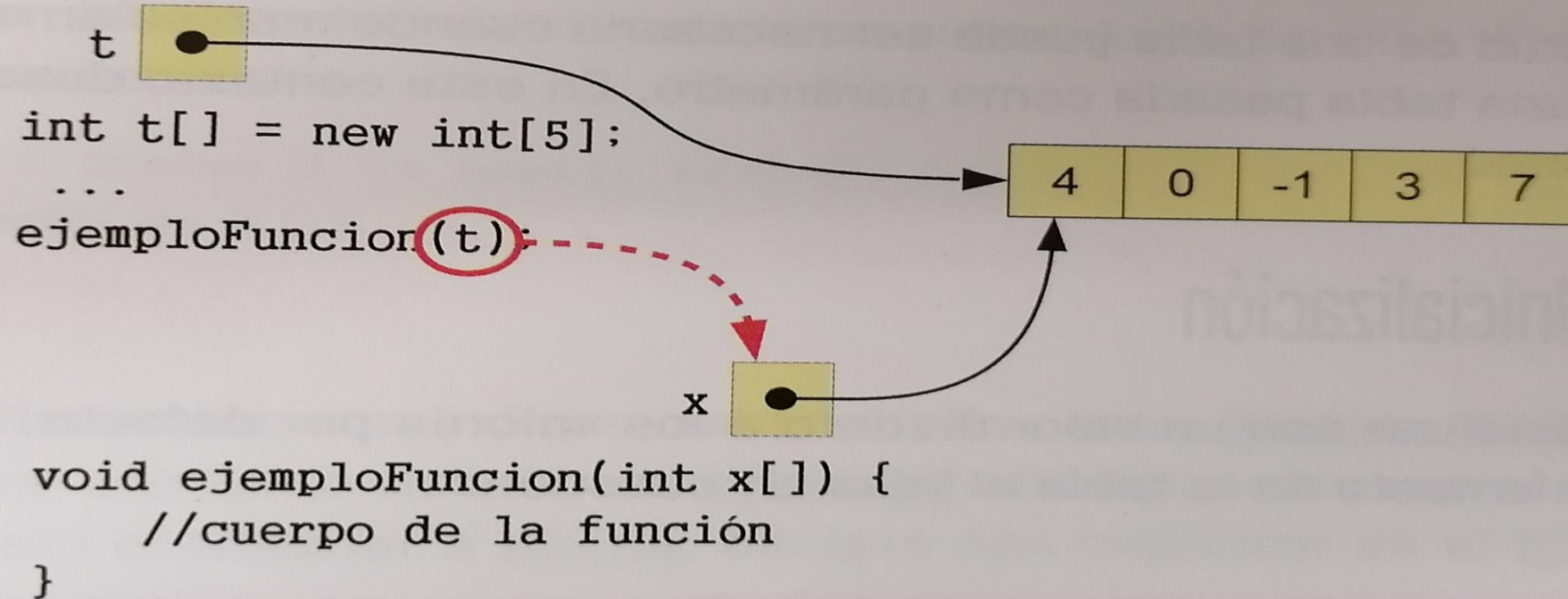


Figura La variable `t`, que referencia a la tabla de datos, copia su referencia a la variable `x` (parámetro de la función). El mecanismo de paso de parámetros es siempre el mismo: el valor de la variable (sea escalar o de tipo tabla) se copia en el parámetro. De esta forma, tanto la variable `t` como `x` referencian a la misma tabla

OPERACIONES CON TABLAS: LA CLASE ARRAYS

La API de Java proporciona herramientas que facilitan el trabajo del programador; hemos usado ya las clases Scanner y Math, entre otras. También disponemos de la clase **Arrays**, que tiene una serie de métodos estáticos para operar con tablas. Se ubica en el paquete **java.util** y tiene que ser importada para poder usarse.

```
import java.util.Arrays;
```


OPERACIONES CON TABLAS: LA CLASE ARRAYS

No existe ningún motivo por el que no podamos implementar nuestras propias operaciones para tablas, pero Arrays nos da la seguridad de un código eficiente, sin errores y la comodidad que supone ahorrarnos el tiempo de escribir nuestra implementación.

Siempre que sea posible aprovecharemos las funcionalidades presentes en la clase Arrays.

Las opciones al trabajar con tablas son prácticamente infinitas, pero casi todo lo que necesitamos puede sintetizarse en las operaciones que veremos a continuación.

OBTENCIÓN DEL NÚMERO DE ELEMENTOS DE UNA TABLA

Java proporciona un mecanismo para conocer el número de elementos, “***longitud***”, con el que se construyó una tabla.

nombreVariable.length

Por ejemplo, el código:

```
int notas[ ] = new int [10];  
System.out.println("Longitud de la tabla notas: " + notas.length);
```

muestra por pantalla:

Longitud de la tabla notas: 10

OBTENCIÓN DEL NÚMERO DE ELEMENTOS DE UNA TABLA

Averiguar la longitud de una tabla puede ser necesario cuando manipulamos, en el cuerpo de una función, una tabla pasada como parámetro. En este contexto desconocemos con qué longitud se creó.

INICIALIZACIÓN

Si deseamos inicializar con un valor distinto a los valores por defecto, tendremos que asignar a cada elemento de la tabla el valor en cuestión.

INICIALIZACIÓN

Recordatorio: Por defecto, una tabla se inicializa con valores específicos:

- *0 para tipos numéricos.*
- *false para booleanos*

Aunque todavía no lo hemos visto (se verá en la unidad 7), ***cuando la tabla contiene objetos las tablas se inicializan a null.***

INICIALIZACIÓN

Existe un método de la clase Arrays que hace exactamente esto.

- ***static void fill (tipo t, tipo valor)***: que inicializa todos los elementos de la tabla t con valor. Esta función está sobrecargada, siendo posible utilizarla con cualquier tipo primitivo, representado por tipo, con la restricción de que el tipo de tabla coincida con el tipo del valor pasado como parámetro.

Veamos cómo inicializar la tabla sueldos con un valor de 1234,56:

Arrays.fill(sueldos, 1234.56); // inicializa todos los elementos

INICIALIZACIÓN

Si interesa inicializar solo algunos elementos de una tabla, disponemos de:

- ***static void fill (tipo t[], int desde, int hasta, tipo valor)***: asigna los elementos de la tabla t, comprendidos entre los índices desde y hasta, sin incluir este último, con valor. tipo representa cualquier tipo primitivo, con la restricción de que el tipo de la tabla coincida con el tipo del valor pasado como parámetro.

INICIALIZACIÓN

Como ejemplo, veamos cómo inicializar los elementos como índices del 3 al 6 (en la llamada utilizaremos 7, ya que no se incluye en el rango) de la tabla sueldos:

Arrays.fill(sueldos, 3, 7, 1234.56); // inicializa solo el rango 3..6

El rango de índices es el comprendido entre el 3 y el anterior al 7, es decir, del 3 al 6.

RECORRIDO

Muchas operaciones con tablas implican recorrerlas, que consiste en visitar sus elementos para procesarlos. Por **procesar** se entiende cualquier operación que realicemos con un elemento, como, por ejemplo, asignarle un valor, mostrarlo por consola o hacer algún tipo de cálculo con él.

El recorrido de una tabla puede ser total, cuando se recorren todos sus elementos o parcial, cuando solo visitamos un subconjunto de ellos.

RECORRIDO

El patrón de código para recorrer una tabla es:

```
for (int i = desde; i <= hasta; i++) {  
  
    // procesado de t[i]  
    ...  
}
```

RECORRIDO

Se visitan los elementos con índices comprendidos entre *desde* y *hasta*. En el código anterior, el último elemento visitado es t[hasta]. Si deseamos que el último elemento visitado sea justo el anterior a hasta, bastará con modificar en el for la condición:
 $i \leq \text{hasta}$ por $i < \text{hasta}$.

Por ejemplo, si deseamos incrementar un 10% todos los elementos de la tabla sueldos,

```
for (int i = 0; i < sueldos.length; i++) { //recorremos la tabla  
    sueldos[ i ] = sueldos[ i ] + 0.1 * sueldos[ i ]; // procesamos  
}
```

RECORRIDO

La instrucción `for` tiene una sintaxis alternativa, conocida como ***for-each*** o ***for extendido***, que permite recorrer los elementos de una tabla.

```
for (declaración variable: tabla) {  
    ...  
}
```

RECORRIDO

Es necesario declarar una variable que tiene que ser del mismo tipo que la tabla. Esta variable irá tomando en cada iteración cada uno de los valores de los elementos de la tabla y el bucle se ejecutará tantas veces como elementos existan.

Es importante tener en cuenta que la variable es una copia de cada elemento, y que en el caso de que se modifique, estamos modificando una copia, no el elemento de la tabla.

RECORRIDO

Veamos cómo sumar todos los elementos de la tabla sueldos.

```
double sumaSueldos = 0;  
for (double sueldo: sueldo) { // sueldo tomará todos los valores de la tabla  
    sumaSueldos += sueldo;  
}
```

MOSTRAR UNA TABLA

Mostrar una tabla consiste en mostrar sus elementos. Si ejecutamos:

```
int t[ ] = {8, 41, 37, 22, 19};
```

```
System.out.println(t); // muestra una referencia  
}
```

No se muestra el contenido de la tabla; en su lugar se muestra la referencia que contiene t.

MOSTRAR UNA TABLA

Para mostrar una tabla tendremos que realizar un recorrido en el que mostrar, uno a uno, cada elemento que la compone. Esta funcionalidad la realiza el método estático **toString()** de la clase **Array**, que se usa en combinación con **System.out.println()**.

Veamos un ejemplo:

```
int t[ ] = {8, 41, 37, 22, 19};  
  
System.out.println(Arrays.toString(t));  
}
```

Muestra los valores de la tabla entre corchetes: **[8, 41, 37, 22, 19]**.

MOSTRAR UNA TABLA

Si preferimos escribir nuestra propia implementación, tendremos que recorrer la tabla y mostrar sus elementos, de la siguiente forma:

```
for (int i = 0; i < t.length; i++) { //recorremos toda la tabla  
    System.out.println(t[ i ]); // mostramos cada elemento  
}
```

o utilizando for-each:

```
for (int elemento: t) {  
    System.out.println(elemento);  
}
```

Evidentemente, es más cómodo utilizar **Arrays.toString()**.

ACTIVIDADES

Actividad 5.1. (135) Crear una tabla de longitud 10 que se inicializará con números aleatorios comprendidos entre 1 y 100. Mostrar la suma de todos los números aleatorios que se guarden en la tabla.

Actividad 5.2. (137) Diseñar un programa que solicite al usuario que introduzca por teclado 5 números decimales. A continuación, mostrar los números en el mismo orden que se han introducido.

ACTIVIDADES

Actividad 5.3. (137) Escribir una aplicación que solicite al usuario cuántos números desea introducir. A continuación, introducir por teclado esa cantidad de números enteros, y por último, mostrar en el orden inverso al introducido.

Actividad 5.4. (138) Diseñar la función: **int maximo(int t[])**, que devuelva el máximo valor contenido en la ***tabla t***.

ORDENACIÓN

Ordenar una tabla consiste en cambiar de posición los datos que contiene para que, en conjunto, resulten ordenados. La clase Arrays permite ordenar tablas mediante el método:

- ***static void sort (tipo t[])***: ordena los elementos de la tabla t de forma creciente. El método se encuentra sobrecargado para cualquier tipo primitivo; de ahí que tipo pueda ser int, double, etcétera.

ORDENACIÓN

Veamos cómo ordenar una tabla:

int edad = {85, 19, 3, 23, 7}; // tabla desordenada

Arrays.sort(edad); // ordena la tabla. Ahora edad = [3, 7, 19, 23, 85]

ORDENACIÓN

Consejo: Buscar en una tabla ordenada es una operación muy rápida; por el contrario, hacerlo en una tabla sin ordenar requiere de mucho tiempo. Sin embargo, el proceso de ordenación es muy largo.

Por ello, antes de ordenar una tabla hay que plantearse si realmente es necesario, y si compensa hacerlo para que las búsquedas sean más rápidas. Solo merecerá la pena ordenar una tabla si vamos a realizar muchas búsquedas en ella.

ACTIVIDADES

Actividad 5.5. (138) Escribir la función `int[] rellenaPares(int longitud, int fin)`, que crea y devuelve una tabla ordenada de la longitud especificada, que se encuentra rellena con números pares aleatorios comprendidos en el rango 2 hasta `fin` (inclusive).

BÚSQUEDA

Consiste en averiguar si entre los elementos de una tabla se encuentra, y en qué posición, un valor determinado llamado **clave de búsqueda**. El algoritmo de búsqueda depende de si la tabla está o no ordenada.

Una búsqueda en una tabla ordenada siempre es más rápida que buscar en la misma tabla con sus elementos no ordenados.

BÚSQUEDA EN UNA TABLA NO ORDENADA

Se denomina ***búsqueda secuencial*** y consiste en un recorrido de la tabla donde se comprueban los valores de los elementos. El proceso finalizará cuando encontremos la clave de búsqueda o cuando no existan más elementos donde buscar.

Dicho de otro modo, mientras no encontremos el valor buscado o el final de la tabla, hemos de continuar con la búsqueda.

BÚSQUEDA EN UNA TABLA NO ORDENADA

El siguiente algoritmo busca **claveBusqueda** en una tabla **t** no ordenada.

```
/* búsqueda secuencial */  
int indiceBusqueda = 0; // índice que usamos para recorrer la tabla  
while (indiceBusqueda < t.length && // no es el último elemento  
       t[indiceBusqueda] != claveBusqueda) { // y no encontrado  
    indiceBusqueda++; // incrementamos el índice de búsqueda  
}  
if (indiceBusqueda < t.length) {  
    ... // claveBusqueda se encuentra en la posición indiceBusqueda  
} else { // el índice se ha salido de rango  
    ... // no encontrado  
}
```

BÚSQUEDA EN UNA TABLA NO ORDENADA

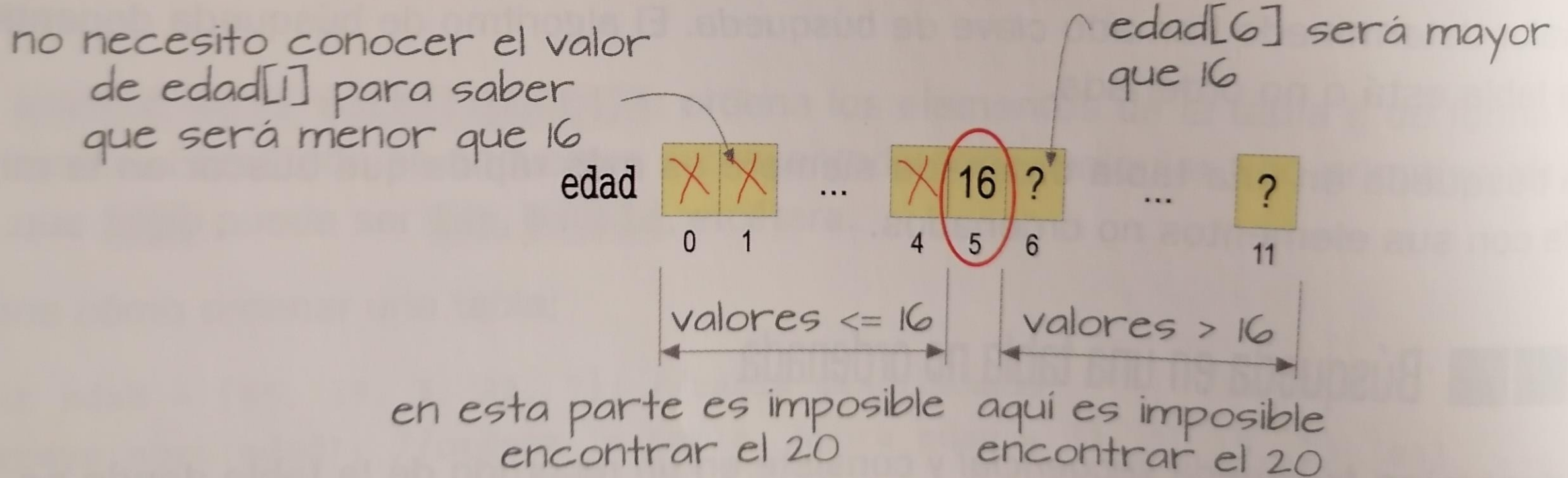
Cuando salimos del bucle while es por dos motivos: o bien hemos encontrado el elemento buscado o, por el contrario, no lo hemos encontrado y no hay más elementos donde buscar.

BÚSQUEDA EN UNA TABLA ORDENADA

Aprovechamos que la posición de los valores nos proporciona información extra. Supongamos que en la tabla edad, con una longitud de 12 y ordenada de forma creciente, buscamos si alguien tiene 20 años. Y sabemos que edad[5] es 16. Sin necesidad de conocer más valores de la tabla, deducimos que:

- Los valores de edad[0] ... edad[4] serán menores o iguales que 16, y aquí es imposible encontrar el 20.
- edad[5] vale 16.
- En el caso de encontrarse, el valor 20 estará a partir de edad[6].

BÚSQUEDA EN UNA TABLA ORDENADA



Figura

Búsqueda dicotómica. El elemento central de una tabla proporciona información extra de dónde buscar.

BÚSQUEDA EN UNA TABLA ORDENADA

En este ejemplo podemos descartar la primera mitad de la tabla, ya que al estar ordenada, es imposible encontrar el valor 20. En caso de existir, estará en la segunda mitad. Este algoritmo se repetirá sucesivas veces, siempre en la mitad donde es posible encontrar el valor.

Este comportamiento hace que sea la forma más eficiente de buscar.

BÚSQUEDA EN UNA TABLA ORDENADA

Esta propiedad de las tablas ordenadas se aprovecha en el algoritmo de **búsqueda dicotómica**, también llamado **búsqueda binaria**, que comprueba si la clave de búsqueda se encuentra en el elemento central de la tabla. Con esta información sabe si debe seguir buscando en la primera o en la segunda mitad de la tabla.

El proceso se repite con la mitad, donde es posible encontrar la clave de búsqueda, que se subdivide de nuevo en dos partes. El algoritmo continúa hasta encontrar la clave de búsqueda o hasta que no existan más elementos donde buscar.

BÚSQUEDA EN UNA TABLA ORDENADA

Podemos escribir nuestro propio algoritmo de búsqueda dicotómica, pero no es necesario, ya que se encuentra implementada en la clase Arrays.

- ***static int binarySearch (tipo t[], tipo claveBusqueda)***: busca de forma dicotómica en la tabla t (que supone ordenada) el elemento con valor claveBusqueda. Devuelve el índice donde se encuentra la primera ocurrencia del elemento buscado o un valor negativo en caso contrario.

BÚSQUEDA EN UNA TABLA ORDENADA

Veamos un ejemplo: deseamos buscar en la tabla ordenada precios, si existe y en qué posición está algún producto de 19,95 euros.

```
int pos = Arrays.binarySearch(precios, 19.95);  
if (pos >= 0) {  
    System.out.println("Encontrado en el índice: " + pos);  
} else {  
    System.out.println("Lo sentimos, no se ha encontrado.");  
}
```

BÚSQUEDA EN UNA TABLA ORDENADA

Aclaración: Cuando el elemento a buscar no se encuentra, el valor negativo devuelto tiene un significado especial: informa de la posición donde tendría que colocarse el elemento buscado para que la tabla continúe ordenada. El índice de inserción se calcula:

$\text{indiceInserción} = -\text{pos} - 1;$

siendo **pos** el valor negativo devuelto por **binarySearch()**.

BÚSQUEDA EN UNA TABLA ORDENADA

Veamos un ejemplo:

```
int a[ ] = {2, 4, 5, 6, 9};
```

```
int pos = Arrays.binarySearch(a, 3); // pos vale -2
```

```
int indiceInsercion = -pos - 1; // vale 1
```

Es decir, si insertamos el valor buscado (3) en la tabla, debemos hacerlo en el índice 1 para que la tabla continúe ordenada.

BÚSQUEDA EN UNA TABLA ORDENADA

El método anterior realiza la búsqueda en toda la tabla, pero si solo interesa buscar en un subconjunto de elementos, disponemos:

- ***static int binarySearch (tipo t[], int desde, int hasta, tipo claveBusqueda)***: solo busca en los elementos comprendidos entre los índices desde y hasta, sin incluir en la búsqueda este último.

BÚSQUEDA EN UNA TABLA ORDENADA

Aclaración: En los métodos de distintas clases de la API, cuando se describe un rango mediante dos índices, **desde** y **hasta**, es habitual que **se incluya en el rango el valor desde y se excluya hasta**.

ACTIVIDADES

Actividad 5.6. (141) Definir una función que tome como parámetros dos tablas, la primera con 6 números de una apuesta de la primitiva, y la segunda (ordenada) con los 6 números de la combinación ganadora. La función devolverá el número de aciertos.

COPIA

El procedimiento para realizar manualmente la copia exacta de una tabla consiste en:

- 1. Crear una nueva tabla, que llamaremos destino o copia, del mismo tipo y longitud que la tabla original.**
- 2. Recorrer la tabla original, copiando el valor de cada elemento en su lugar correspondiente en la tabla destino.**

COPIA

Sin embargo, **Arrays** proporciona esta funcionalidad mediante:

- ***static tipo[] copyOf (tipo origen[], int longitud)***: construye y devuelve una copia de origen con la longitud especificada. Si la longitud de la nueva tabla es menor que la de la original, solo se copian los elementos que caben. En caso contrario, los elementos extras se inicializan por defecto. Este método, como la mayoría de los métodos de Arrays, está sobrecargado para poder trabajar con todos los tipos.

COPIA

Veamos un ejemplo:

```
int t[ ] = {1, 2, 1, 6, 23}; // tabla origen
```

```
int a[ ], b[ ]; // tabla destino
```

```
a = Arrays.copyOf(t, 3); // a = [1, 2, 1]
```

```
b = Arrays.copyOf(t, 10); // b = [1, 2, 1, 6, 23, 0, 0, 0, 0, 0]
```

COPIA

Existe otro método que también realiza una copia de una tabla, pero en este caso de un rango de elementos:

- ***static tipo[] copyOfRange (tipo origen[], int desde, int hasta):*** crea y devuelve una tabla donde se han copiado elementos de origen comprendidos entre los índices desde y hasta, sin incluir este último.

COPIA

Un ejemplo:

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

```
int a[ ] = Arrays.copyOfRange(t, 1, 4); // a = [5, 3, 1]
```

que realiza una copia desde los índices 1 al 3 (el anterior al 4).

COPIA

Otro método disponible es **arraycopy()** de la clase **System**, que copia elementos consecutivos entre dos tablas. La diferencia entre **arraycopy()** y **copyOfRange()** es que el primero no crea ninguna tabla, ambas tablas deben estar creadas previamente. Su sintaxis es:

- ***void arraycopy (Object tablaOrigen, int posOrigen, Object tablaDestino, int posDestino, int longitud)***: copia en la *tablaDestino*, a partir del índice *posDestino*, los datos de la *tablaOrigen*, comenzando en el índice *posOrigen*. El parámetro *longitud* especifica el número de elementos que se copiarán entre ambas tablas. Hay que tener precaución, ya que los valores de los elementos afectados por la copia de la tabla destino se perderán.

COPIA

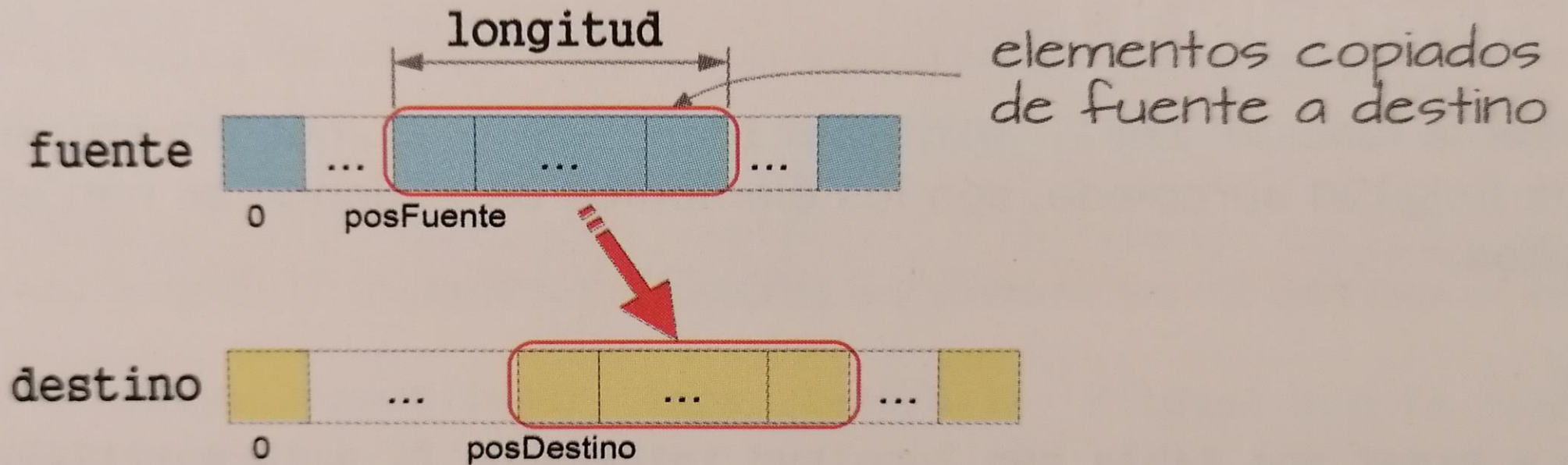


Figura Proceso de copia que realiza `copyarray()`. Los elementos copiados pueden moverse desde su posición en la tabla fuente (u origen) a cualquier otra posición en la tabla destino. Ambas tablas pueden ser de distinta longitud, aunque siempre hay que estar seguro de que no copiaremos en elementos fuera de rango, lo que produciría un error.

COPIA

Aclaración: Object es una forma de llamar en Java a cualquier cosa, incluidas las tablas. Por ahora no estamos trabajando con clases, pero pronto empezaremos a ver programación orientada a objetos.

INSERCIÓN

La forma de añadir un nuevo valor a una tabla depende de si está o no ordenada.

Si el orden no importa, basta con incrementar la longitud de la tabla e insertar el nuevo dato en el último elemento.

Y cuando la tabla está ordenada, hemos de insertar el nuevo dato de forma que todos los valores sigan ordenados.

INSERCIÓN NO ORDENADA

Veamos el algoritmo para insertar el valor nuevo, en un elemento que añadimos al final de la tabla `t`. Hay que destacar que la longitud de la tabla no se modifica; lo que realmente ocurre es que se crea una segunda tabla (copia de la tabla original) en la que hemos aumentado la longitud en uno.

La nueva tabla se referencia con la misma variable `t`, dando la sensación de que la hemos modificado. La tabla original, al quedar sin referencia, queda a merced del recolector de basura.

INSERCIÓN NO ORDENADA

La siguiente figura representa lo que ocurre en el siguiente código:

```
t = Arrays.copyOf(t, t.length + 1);  
t[t.length - 1] = nuevo;
```

la copia incrementa la longitud

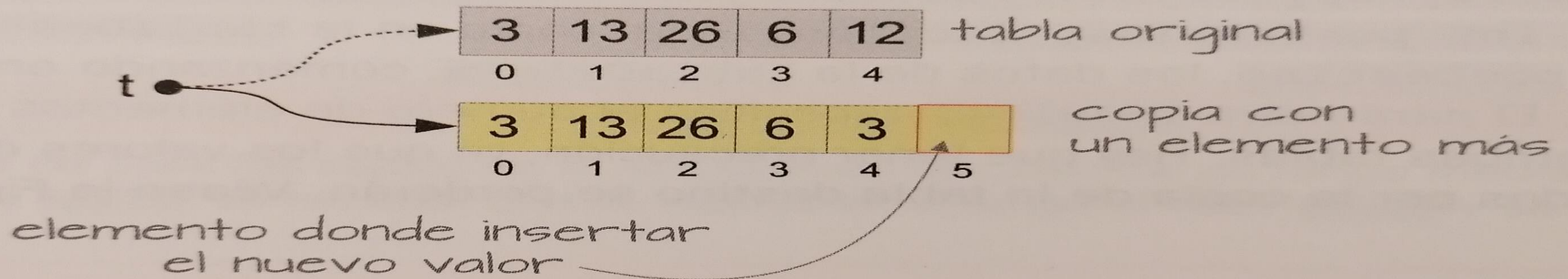


Figura Inicialmente la tabla original estaba referenciada por `t`. Tras ejecutar `Arrays.copyOf()`, `t` se modifica y pasa a referenciar a la nueva tabla, que es una copia con una longitud incrementada en uno. Ahora disponemos de un nuevo elemento para añadir un dato más.

ACTIVIDADES

Actividad 5.7. (144) Implementar la función `int[] sinRepetidos(int t[])`, que construye y devuelve una tabla de la longitud apropiada, con los elementos de `t`, donde se han eliminados los datos repetidos.

Actividad 5.8. (144) Leer y almacenar n números enteros en una tabla, a partir de la que se construirán otras dos tablas con los elementos con valores pares e impares de la primera, respectivamente. Las tablas pares e impares deben mostrarse ordenadas.

INSERCIÓN ORDENADA

La inserción ordenada consiste en añadir un nuevo elemento en la tabla en la posición adecuada para que la tabla continúe ordenada.

Primeramente, buscaremos el lugar que le correspondería al nuevo valor en la tabla; a este índice le llamaremos `indiceInsercion`. A continuación, crearemos una nueva tabla, que llamaremos `copia`, con un elemento extra.

INSERCIÓN ORDENADA

Ahora hemos de copiar los elementos de la tabla original a la tabla copia, teniendo la precaución de no utilizar el elemento situado en `indiceInsercion`, que es un hueco que está reservado para el nuevo valor.

Es decir, todos los elementos cuyos índices son anteriores a `indiceInsercion` se copian en la misma posición, y los posteriores, se copian desplazados un elemento hacia el final de la tabla. Con esto conseguimos que, tras insertar el nuevo valor en el elemento marcado por `indiceInsercion`, la tabla se mantenga ordenada.

INSERCIÓN ORDENADA

Finalmente, la copia será referenciada por la misma variable que referenciaba la tabla original, dando la sensación de que la tabla ha crecido. Veamos un ejemplo:

```
int t[ ] = {1, 2, 3, 4, 6, 7, 8};
int nuevo = 5;
int indiceInsercion = Arrays.binarySearch(t, nuevo);
// si indiceInsercion >= 0, el nuevo elemento (que está repetido) se inserta en
// en el lugar en que ya estaba, desplazando al original. Si por el contrario:
if (indiceInsercion < 0) { // si no lo encuentra
    // calcula donde debería estar
    indiceInsercion = -indiceInsercion - 1;
}
```

INSERCIÓN ORDENADA

```
int copia[ ] = new int [t.length + 1]; // nueva tabla con longitud + 1
// copiamos los elementos antes del "hueco"
System.arraycopy(t, 0, copia, 0, indiceInsercion);
// copiamos desplazados los elementos tras el "hueco"
System.arraycopy(t,   indiceInsercion,   copia,   indiceInsercion   +   1,
t.length - indiceInsercion);
copia[ indiceInsercion ] = nuevo; // asignamos el nuevo elemento
t = copia; // t referencia la nueva tabla
System.out.println(Arrays.toString(t)); // mostramos
```

INSERCIÓN ORDENADA

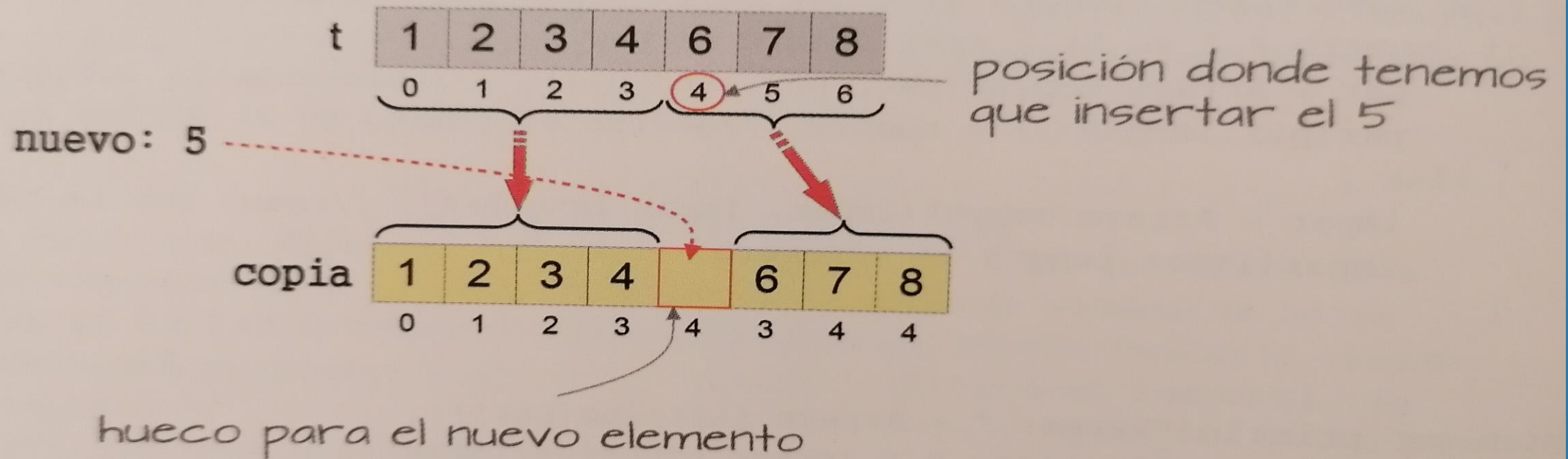


Figura Momento en el que hemos creado el hueco para el nuevo elemento y hemos copiado los datos. Solo queda insertar el nuevo elemento (5) en el hueco y referenciar la tabla **copia** con **t**, dando la sensación de que la tabla ha incrementado su longitud. Es importante que, tras todas las operaciones, la tabla **t** continúe estando ordenada.

ACTIVIDADES

Actividad 5.9. (146) Diseñar una aplicación para gestionar un campamento de programación, donde se introduce la puntuación (enteros) obtenidos por 5 programadores, conforme van terminando su prueba. La aplicación debe mostrar las puntuaciones ordenadas de los 5 participantes. En ocasiones, cuando finalizan los 5 participantes anteriores, se suman al campeonato programadores de exhibición, cuyos puntos se incluyen con el resto. La forma de especificar que no intervienen más programadores de exhibición es introducir como puntuación un -1.

La aplicación debe mostrar, finalmente, los puntos ordenados de todos los participantes.

ELIMINACIÓN

Esta operación consiste en borrar un elemento de la tabla, por lo que después de una eliminación la longitud de la tabla decrece.

Antes de eliminar un elemento de una tabla, siempre tendremos que buscarlo para conocer en qué índice se encuentra.

Una vez localizado, la operación dependerá del tipo de tabla con la que estemos trabajando.

ELIMINACIÓN. Tabla no ordenada

Para eliminar un elemento en una tabla, después de buscarlo, lo sustituimos por el último de la tabla (que ahora estará repetido). A continuación, creamos una copia de la tabla con los mismos datos, pero disminuyendo su longitud, lo que provoca que perdamos el último elemento, que es el que estaba repetido.

Veamos el algoritmo donde $t[]$ es la tabla con los datos e `indiceBorrado` contendrá, si existe, el índice del elemento que deseamos eliminar, que se almacena en la variable `aBorrar`:

ELIMINACIÓN. Tabla no ordenada

```
... // algoritmo de búsqueda, que devuelve el índice del elemento a borrar si
    // existe, o -1 si no existe
if (indiceBorrado != -1) { // encontrado
    t [indiceBorrado] = t [t.length - 1]; // copia el último en indiceBorrado
    t = Arrays.copyOf (t, t.length - 1); // disminuimos la longitud de t
    System.out.println(Arrays.toString(t)); // mostramos
} else {
    ... // no podemos borrar nada, ya que no lo hemos encontrado
}
```

ELIMINACIÓN. Tabla no ordenada

aBorrar: 46

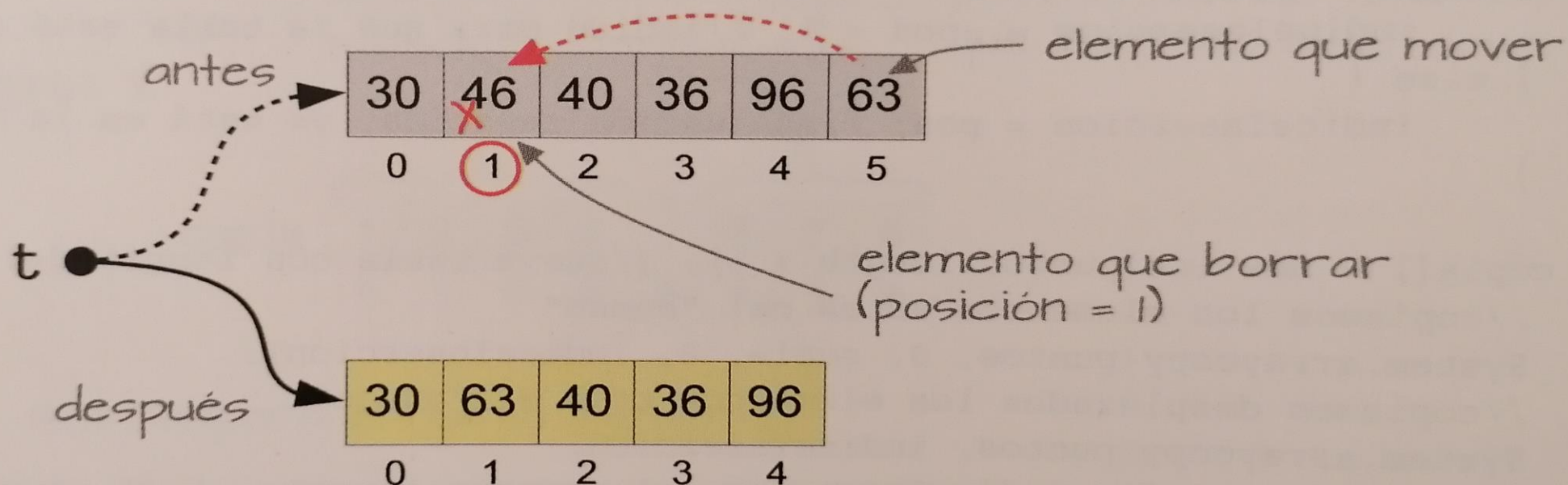


Figura Todas las operaciones de eliminación comienzan localizando el elemento que se va a borrar. Después se han de recolocar el resto de elementos para que produzcan el efecto de que el elemento en cuestión ha desaparecido. Finalmente, redimensionamos a una tabla más pequeña.

ACTIVIDADES

Actividad 5.10. (148) Escribir la función `int[] eliminarMayores(int t[], int valor)` que crea y devuelve una copia de la tabla `t` donde se han eliminado todos los elementos que son mayores que `valor`.

ELIMINACIÓN. Tablas ordenadas

El primer paso es buscar el elemento que se va a borrar (variable aBorrar). Al estar la tabla ordenada podemos utilizar la búsqueda dicotómica. Este algoritmo busca el índice (variable indiceBorrado) que utilizaremos para determinar el elemento que eliminar.

En tablas ordenadas, al eliminar un elemento tenemos que seguir manteniendo los demás valores contiguos y en el mismo orden. Para ello, tenemos que desplazar los valores que siguen a indiceBorrado una posición hacia el principio. Con esta técnica sobrescribimos el valor que borrar y obtenemos un hueco libre al final de la tabla, que desaparecerá al disminuir su longitud.

ELIMINACIÓN. Tablas ordenadas

aBorrar: 33

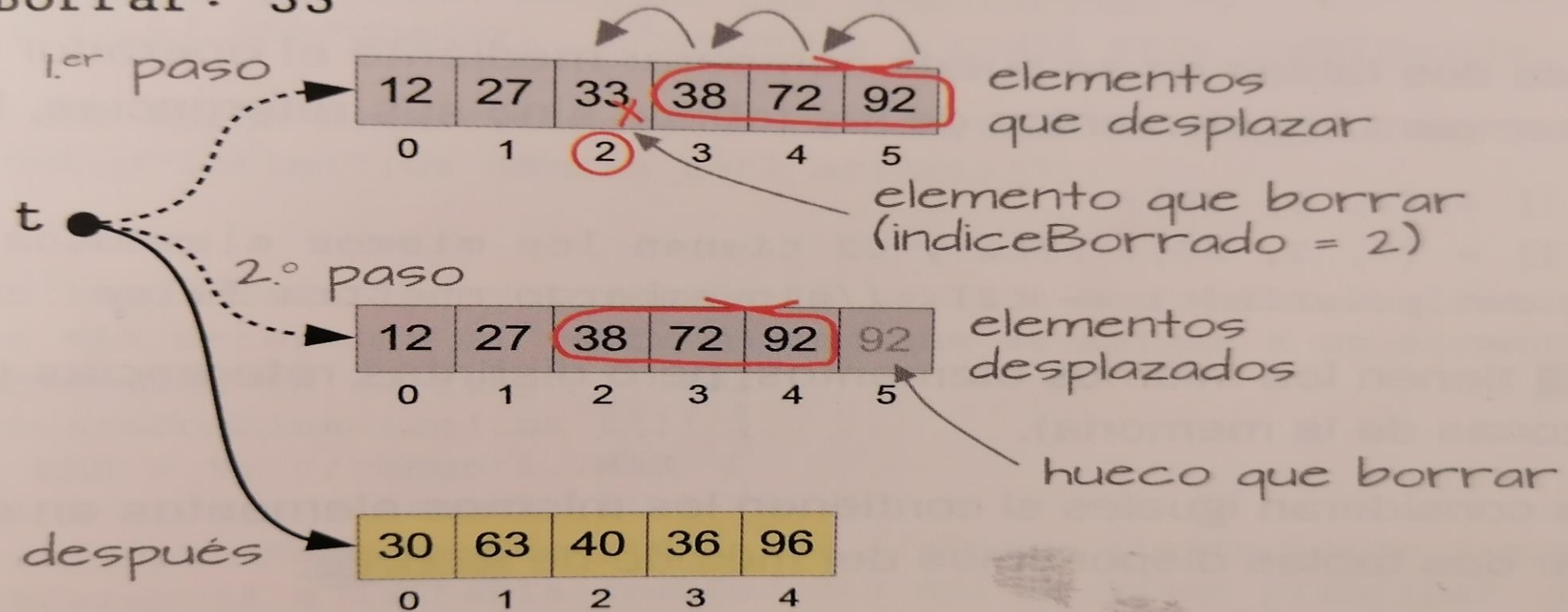


Figura El borrado en una tabla ordenada implica un primer paso donde se localiza el elemento que borrar, un segundo paso donde se desplazan los elementos a la derecha del elemento que nos interesa y, finalmente, el redimensionado de la tabla.

ELIMINACIÓN. Tablas ordenadas

Veamos a modo de ejemplo cómo eliminar un elemento que se solicita por teclado:

```
int t[ ] = {12, 27, 33, 38, 72, 92};
int aBorrar = new Scanner(System.in).nextInt( );
// usamos el algoritmo de búsqueda dicotómica
int indiceBorrado = Arrays.binarySearch(t, aBorrar);
if (indiceBorrado >= 0) {
    // desplazamos los elementos posteriores a indiceBorrado
    System.arraycopy(t, indiceBorrado + 1, t, indiceBorrado, t.length - indiceBorrado - 1);
    t = Arrays.copyOf (t, t.length - 1); // disminuimos la longitud
    System.out.println(Arrays.toString(t)); // mostramos
} else {
    ... // no podemos borrar nada, ya que no lo hemos encontrado
}
```


COMPARACIÓN DE DOS TABLAS

El contenido de dos tablas no se puede comparar mediante el operador `==`, ya que este operador no compara los elementos de las tablas, sino sus referencias. Por ejemplo:

```
int t1[ ] = {7, 9, 20};
```

```
int t2[ ] = {7, 9, 20}; // t1 y t2 tienen los mismos elementos
```

```
System.out.println(t1 == t2); // sin embargo muestra false
```

ya que t1 y t2 tienen los mismos elementos, pero distintas referencias (están ubicadas en distintas zonas de memoria).

COMPARACIÓN DE DOS TABLAS

Dos tablas se consideran iguales si contienen los mismos elementos en el mismo orden. Para comparar dos tablas disponemos del método de Arrays.

- ***static boolean equals(tipo a[], tipo b[])***: compara las tablas a y b, elemento a elemento. En el caso de que sean **iguales** devuelve **true**, y en caso contrario, **false**.

Veamos cómo comparar las dos tablas anteriores:

```
System.out.println(Arrays.equals(t1, t2)); // muestra true
```

ACTIVIDADES

Actividad 5.11. (151) Desarrollar el juego “La cámara secreta”, que consiste en abrir una cámara mediante una combinación secreta, que está formada por una combinación de dígitos del uno al cinco. El jugador especificará cual es la longitud de la combinación; a mayor longitud, mayor será la dificultad del juego. La aplicación genera, de forma aleatoria una combinación secreta que el usuario tendrá que acertar. En cada intento se muestra como pista, para cada dígito de la combinación introducido por el jugador, si es mayor, menor o igual que el correspondiente en la combinación secreta.

TABLAS N-DIMENSIONALES

Hasta el momento las tablas que hemos utilizado han sido unidimensionales: solo tienen longitud; dicho de otra forma, los elementos solo se extienden a lo largo de un eje (el eje X), y basta con un índice para recorrerlas.

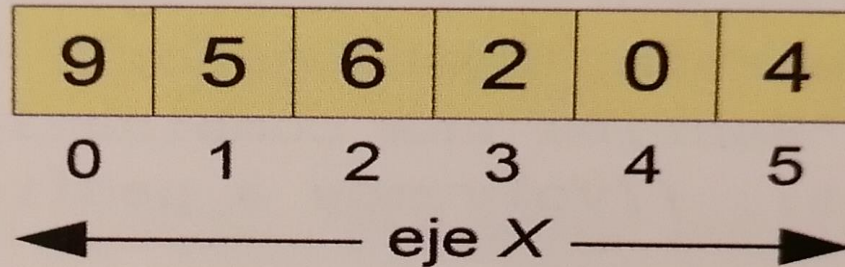


Figura. El eje X, o eje de abscisa, es el eje horizontal. Hay que entender que esto es una representación, y lo habitual es que nos figuremos que las tablas unidimensionales se expanden horizontalmente.

TABLAS N-DIMENSIONALES

Pero puede ocurrir que nuestros datos se refieran a entidades que se caracterizan por más de una propiedad, de las cuales alguna o algunas sirven para identificarlo. En este caso, no nos basta con un solo índice para describirlas.

TABLAS BIDIMENSIONALES

Podemos ampliar el concepto de tabla haciendo que los elementos se extiendan en dos dimensiones, utilizando los ejes X e Y. Ahora la tabla posee longitud y anchura. Para identificar cada elemento de una tabla unidimensional hemos utilizado un índice; para las tablas bidimensionales, compuestas por filas y columnas, se necesita un par de índices $[x] [y]$. Una tabla bidimensional recibe el nombre de matriz, aunque a diferencia de las matemáticas, en Java se numeran comenzando por 0.

TABLAS BIDIMENSIONALES

La declaración de una tabla bidimensional se hace de la forma:

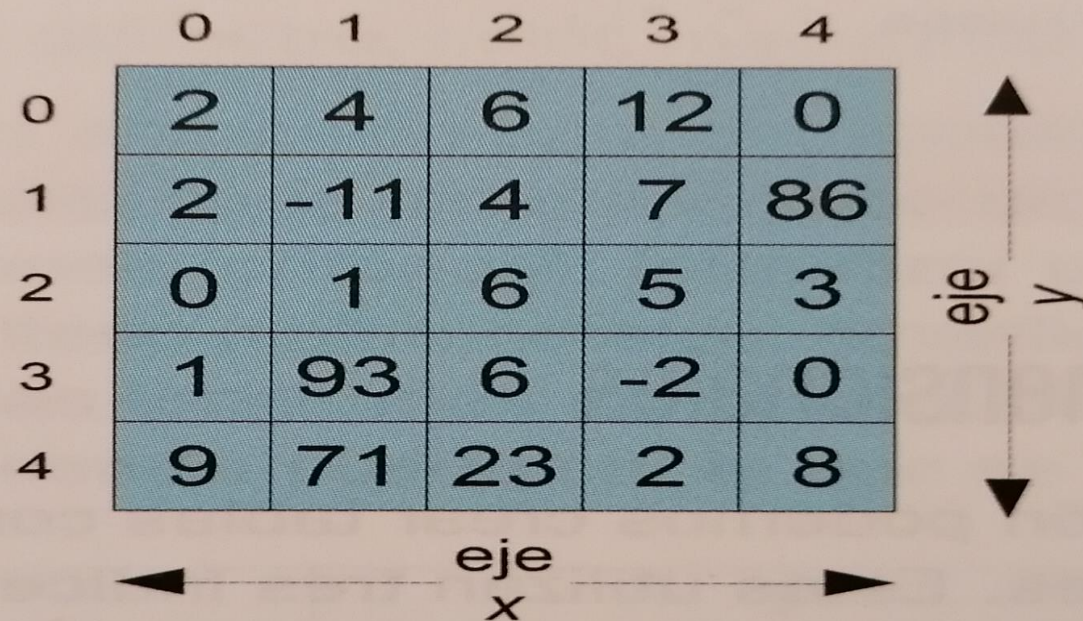
tipo nombreTabla[] [];

A continuación, creamos la tabla, indicando la longitud de cada dimensión. Veamos cómo crear la tabla datos que se corresponde con la siguiente figura:

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


TABLAS BIDIMENSIONALES

y con ello, estamos reservando espacio en la memoria para (5 x 5) 25 elementos.



A 5x5 matrix is shown with its elements. The columns are indexed 0 to 4 from left to right, and the rows are indexed 0 to 4 from top to bottom. The matrix is labeled with 'eje x' for the horizontal axis and 'eje y' for the vertical axis. The elements are as follows:

	0	1	2	3	4
0	2	4	6	12	0
1	2	-11	4	7	86
2	0	1	6	5	3
3	1	93	6	-2	0
4	9	71	23	2	8

Figura Matriz de 5×5 elementos. Ahora la identificación de cada elemento viene dada por el índice del eje X y el índice del eje Y.

TABLAS BIDIMENSIONALES

Los algoritmos que utilizan matrices requieren dos bucles anidados. Un bucle se encarga del índice para la dimensión X y el otro para el índice del eje Y. Veamos un ejemplo para introducir por teclado la matriz datos:

```
for (i = 0; i < 5; i++) { // eje X
    for (j = 0; j < 5; j++) { // eje Y
        datos [ i ] [ j ] = sc.nextInt( ); // leemos el elemento [ i ] [ j ]
    }
}
```

TABLAS BIDIMENSIONALES

Para mostrar una tabla bidimensional podemos usar dos bucles anidados como los anteriores o bien utilizar el método estático **Arrays.deepToString()**. Por ejemplo, para mostrar la tabla datos.

```
System.out.println(Arrays.deepToString(datos));
```

ACTIVIDADES

Actividad 5.12. (153) Crear una tabla bidimensional de 5x5 y rellenarla de la siguiente forma: el elemento de la posición $[n][m]$ debe contener el valor $10 * n + m$. Después se debe mostrar su contenido.

TABLAS TRIDIMENSIONALES

Añadiendo una nueva dimensión podemos crear tablas con anchura, altura y profundidad, es decir, tablas tridimensionales. Estas utilizan tres índices ($[x]$ $[y]$ $[z]$) para identificar cada elemento que la componen.

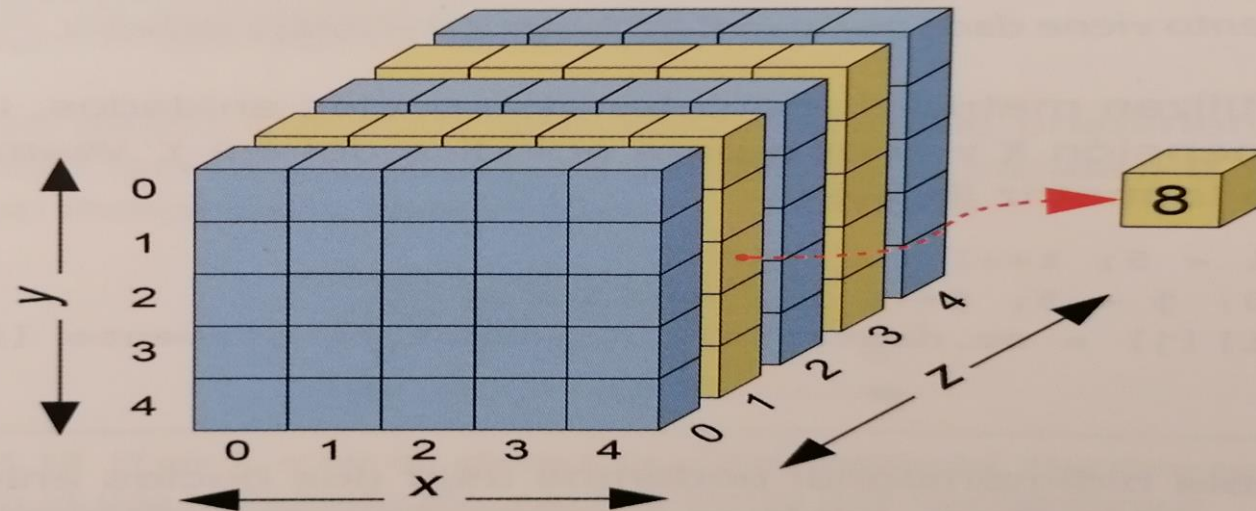


Figura Una tabla tridimensional se representa mediante un cubo. Se aprecia cómo se necesitan tres dimensiones (tres índices) para poder identificar cualquier elemento, que contiene un dato del tipo del que está declarada la tabla. En nuestro ejemplo, el índice $[4][2][1]$ (que corresponde a: x, y, z) vale 8.

TABLAS CON MÁS DIMENSIONES

Dibujar o imaginar una tabla de más de tres dimensiones es algo complicado. Nosotros vivimos en un mundo 4-dimensional, con tres dimensiones para localizar cada objeto en el espacio y una cuarta dimensión, el tiempo, para ver la evolución de un objeto en movimiento.

Pero más allá de nuestro mundo, es complicado representar, o siquiera imaginar, una tabla multidimensional. Un truco sencillo consiste en descomponer la tabla en otras más simples.

TABLAS CON MÁS DIMENSIONES

Por ejemplo, una tabla de 5 dimensiones puede verse como una tabla tridimensional, donde en cada elemento de la tabla se almacena una tabla bidimensional. De los cinco índices necesarios para identificar los elementos de una tabla 5-dimensional, podemos utilizar los tres primeros en la tabla tridimensional y utilizar los otros dos índices para situarnos en la segunda tabla bidimensional.

TABLAS CON MÁS DIMENSIONES

Pero el hecho de que no podamos dibujarla ni imaginarla no significa que no se puedan manipular sus elementos.

Las tablas n-dimensionales son útiles para manejar la información atendiendo a criterios de clasificación. No es necesario que la información tenga una representación gráfica. Veamos un ejemplo: supongamos una máquina que procesa naranjas y necesitamos, por motivos de calidad, clasificarlas y llevar la cuenta del número de frutas recogidas de cada tipo, atendiendo a los criterios: diámetro, color, maduración, forma y peso.

TABLAS CON MÁS DIMENSIONES

Al utilizar cinco criterios, lo más apropiado para almacenar los datos es una matriz 5-dimensional, haciendo corresponder cada dimensión con un criterio de clasificación. Falta, para cada una de las dimensiones (criterios), formalizar una correspondencia entre los posibles valores reales de un criterio (color: naranja, amarillo o verde; nivel de maduración: madura o inmadura; etc) con las longitudes de cada dimensión, que utilizaremos como índices. Una posible correspondencia puede ser la que se muestra en la siguiente figura:

TABLAS CON MÁS DIMENSIONES

Diámetro	0	1		2	
	Pequeño, < 4 cm	Mediano, entre 4 y 8 cm		Grande, > 8 cm	

Color	0	1	2
	Naranja	Amarillo	Verde

Maduración	0	1	2	4
	Pasada	Óptima	Algo inmadura	Totalmente inmadura

Forma	0	1
	Redondeada	Otra forma

Peso	0	1	2	3	4	5
	<100 g	100-200 g	200-300 g	300-400 g	400-500 g	>500 g

Figura Correspondencia entre los valores de cada dimensión y clasificaciones de distintos criterios de las naranjas.

TABLAS CON MÁS DIMENSIONES

Crearemos la variable **naranjas**, que será una matriz con 5 dimensiones, donde cada una de ellas representa un criterio: **naranjas[diámetro] [color] [maduración] [forma] [peso]**.

La declaración y creación de la variable es:

```
int naranjas [ ] [ ] [ ] [ ] [ ];  
naranjas = new int [3] [3] [5] [2] [6];
```

TABLAS CON MÁS DIMENSIONES

Si la máquina contabiliza 25 naranjas con:

- Diámetro de 11 cm: primer índice 2.
- De un color naranja intenso: segundo índice 0.
- En su punto óptimo de maduración: tercer índice 1.
- La forma es redonda: cuarto índice 0.
- Pesa 385 g: quinto índice 3.

Haremos la asignación:

naranjas [2] [0] [1] [0] [3] = 25;

ACTIVIDADES RESUELTAS

Actividad 5.1. Crear una tabla de longitud 10 que se inicializará con números aleatorios comprendidos entre 1 y 100. Mostrar la suma de todos los números aleatorios que se guarden en la tabla.

Actividad 5.2. Diseñar un programa que solicite al usuario que introduzca por teclado 5 números decimales. A continuación, mostrar los números en el mismo orden que se han introducido.

ACTIVIDADES

Actividad 5.3. Escribir una aplicación que solicite al usuario cuántos números desea introducir. A continuación, introducir por teclado esa cantidad de números enteros, y por último, mostrar en el orden inverso al introducido.

Actividad 5.4. Diseñar la función: **int maximo(int t[])**, que devuelva el máximo valor contenido en la **tabla t**.

Actividad 5.5. Escribir la función **int [] rellena(int longitud, int fin)**, que crea y devuelve una tabla ordenada de la longitud específica, que se encuentra rellena con número pares aleatorios comprendidos en el rango desde 2 hasta fin (inclusive).

ACTIVIDADES

Actividad 5.6. Definir una función que tome como parámetros dos tablas, la primera con los 6 números de una apuesta de la primitiva, y la segunda (ordenada) con los 6 números de la combinación ganadora. La función devolverá el número de aciertos.

Actividad 5.7. Implementar la función: **int [] sinRepetidos (int t[])**, que construye y devuelve una tabla de la longitud apropiada, con los elementos de **t**, donde se han eliminado los datos repetidos.

ACTIVIDADES

Actividad 5.8. Leer y almacenar n números enteros en una tabla, a partir de la que se construirán otras dos tablas con los elementos con valores pares e impares de la primera, respectivamente. Las tablas pares e impares deben mostrarse ordenadas.

ACTIVIDADES

Actividad 5.9. Diseñar una aplicación para gestionar un campeonato de programación, donde se introduce la puntuación (enteros) obtenidos por 5 programadores, conforme van terminando su prueba. La aplicación debe mostrar las puntuaciones ordenadas de los 5 participantes. En ocasiones, cuando finalizan los 5 participantes anteriores, se suman al campeonato programadores de exhibición, cuyos puntos se incluyen con el resto. La forma de especificar que no intervienen más programadores de exhibición es introducir como puntuación un -1. La aplicación debe mostrar, finalmente, los puntos ordenados de todos los participantes.

ACTIVIDADES

Actividad 5.10. Escribir la función:

```
int [ ] eliminarMayores(int t[ ], int valor)
```

que crea y devuelve una copia de la tabla t donde se han eliminado todos los elementos que son mayores que valor

ACTIVIDADES

Actividad 5.11. Desarrollar el juego “la cámara secreta”, que consiste en abrir una cámara mediante su combinación secreta, que está formado por una combinación de dígitos del uno al cinco.

El jugador especificará cuál es la longitud de la combinación; a mayor longitud, mayor será la dificultad del juego. La aplicación genera, de forma aleatoria, una combinación secreta que el usuario tendrá que acertar. En cada intento se muestra como pista, para cada dígito de la combinación introducido por el jugador, si es mayor, menor o igual que el correspondiente en la combinación secreta.

ACTIVIDADES

Actividad 5.12. Crear una tabla bidimensional de longitud 5 x 5 y rellenarla de la siguiente forma: el elemento de la posición $[n][m]$ debe contener el valor $10 \times n + m$. Después se debe mostrar su contenido.