PROGRAMACIÓN UD-5

TABLAS

ÍNDICE

[INTRODUCCIÓN 4](#_Toc119127171)

[VARIABLES ESCALARES VS TABLAS 4](#_Toc119127172)

[ÍNDICES 5](#_Toc119127173)

[A. ÍNDICES FUERA DE RANGO 5](#_Toc119127174)

[CONSTRUCCIÓN DE TABLAS 5](#_Toc119127175)

[A. LONGITUD Y TIPO 6](#_Toc119127176)

[B. VARIABLES DE TABLA 6](#_Toc119127177)

[C. OPERADOR new 6](#_Toc119127178)

[REFERENCIAS 7](#_Toc119127179)

[A. RECOLECTOR DE BASURA 8](#_Toc119127180)

[B. REFERENCIA NULL 8](#_Toc119127181)

[USO DE TABLAS 8](#_Toc119127182)

[A. TABLAS ORDENADAS 9](#_Toc119127183)

[B. TABLAS + INDICADOR 9](#_Toc119127184)

[TABLAS COMO PARÁMETROS DE FUNCIONES 10](#_Toc119127185)

[OPERACIONES CON TABLAS: LA CLASE ARRAYS 10](#_Toc119127186)

[A. OBTENCIÓN DEL NÚMERO DE ELEMENTOS DE UNA TABLA 10](#_Toc119127187)

[B. INICIALIZACIÓN 10](#_Toc119127188)

[C. RECORRIDO 10](#_Toc119127189)

[D. MOSTRAR UNA TABLA 10](#_Toc119127190)

[E. ORDENACIÓN 10](#_Toc119127191)

[F. BÚSQUEDA 10](#_Toc119127192)

[1. BÚSQUEDA EN UNA TABLA NO ORDENADA 10](#_Toc119127193)

[2. BÚSQUEDA EN UNA TABLA ORDENADA 10](#_Toc119127194)

[G. COPIA 10](#_Toc119127195)

[H. INSERCIÓN 10](#_Toc119127196)

[1. INSERCIÓN NO ORDENADA 10](#_Toc119127197)

[2. INSERCIÓN ORDENADA 10](#_Toc119127198)

[I. ELIMINACIÓN 10](#_Toc119127199)

[1. INSERCIÓN NO ORDENADA 10](#_Toc119127200)

[2. INSERCIÓN ORDENADA 10](#_Toc119127201)

[J. COMPRACIÓN DE DOS TABLAS 10](#_Toc119127202)

[TABLAS n-DIMENSIONALES 10](#_Toc119127203)

[A. TABLAS BIDIMENSIONALES 10](#_Toc119127204)

[B. TABLAS TRIDIMENSIONALES 10](#_Toc119127205)

[C. TABLAS CON MÁS DIMENSIONES 10](#_Toc119127206)

# INTRODUCCIÓN

Una variable sólo puede almacenar un valor simultáneamente en cada instante de tiempo. Esto se denomina variable escalar.

Pero tenemos una forma de almacenar más de un valor de forma simultánea en una variable, es mediante el uso de tablas las cuales reciben también el nombre de vectores o *arrays*.

Tenemos diferentes tipos de datos:

* Literales: Que son los datos que se introducen entrecomillados.
* Constantes: Que son datos que no van a cambiar a lo largo del proceso.
* Variables: Que son datos que sí van a cambiar.
  + Primitivos o simples
    - Short, byte, int y long.
    - Float, double
    - Char
    - Boolean
  + Compuestos
    - Estáticas
      * String
      * Vectores
    - Dinámicas (colecciones)
      * Listas
      * Pilas
      * Colas
      * Árboles
      * Grafos

# VARIABLES ESCALARES VS TABLAS

Podemos definir que una tabla es una variable que permite guardar más de un valor simultáneamente. Podríamos ver una tabla como una variable que es capaz de englobar a otras variables a las cuales llamaremos elementos o componentes referidas con un mismo nombre. Para que esto sea posible todas las variables deben ser del mismo tipo.

El uso de tablas está muy recomendado cuando debemos usar varios datos del mismo tipo de forma simultánea, por ejemplo, si tenemos que guardar 100 edades es mejor crear una tabla para almacenar todas las edades en lugar de crear 100 variables escalares (1 para cada valor).

Además, si de inicio no sabemos cuántos valores vamos a tener que guardar no nos quedará otra opción que usar tablas. Por ejemplo, un programa donde al inicio le preguntemos al usuario cuántas edades va a querer guardar, ese número N de edades no será un número fijo, sino que variará en cada uso del programa.

# ÍNDICES

Es la forma de distinguir entre cada uno de los elementos o componentes que constituyen una tabla. Son una serie de números que se asignan en un orden para cada elemento comenzando por 0 y terminando por N-1 (si la tabla tiene 5 elementos, el último índice será el 4).

Un reloj digital

Descripción generada automáticamente con confianza media

La forma de utilizar un elemento concreto de una tabla es por medio del nombre de la variable que identificará a la tabla junto al índice entre corchetes que distingue a ese elemento.

Por ejemplo, tenemos 4 edades en un array:

* Edad[0]. Sería el primer valor.
* Edad[1]. Sería el segundo valor.
* Edad[2]. Sería el tercer valor.
* Edad[3]. Sería el último valor.

## ÍNDICES FUERA DE RANGO

Un índice fuera de rango es aquel que no existe, por ejemplo, antes hemos comentado un array con 4 edades yendo estas del 0 al 3, si pedimos la Edad[4] el programa será incapaz de encontrarla porque está fuera de rango, lo mismo si pedimos la Edad[-1]. En estos casos se obtendrá un error de tiempo de ejecución que provoca que el programa termine de forma inesperada.

# CONSTRUCCIÓN DE TABLAS

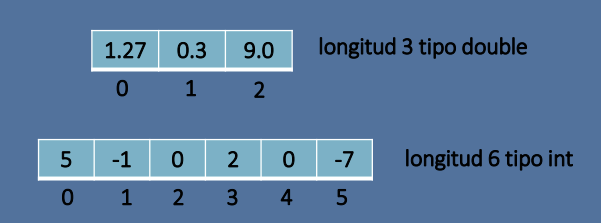
En el momento de creación de la tabla tenemos que:

* 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 por dos elementos:

* + Longitud. Número de elementos que tiene.
  + Tipo. Tipo de los datos que almacena (debe ser el mismo en todos los datos como se ha dicho más arriba).



## VARIABLES DE TABLA

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza mediaLo primero que habrá que declarar es la variable utilizando corchetes, símbolo que diferencia entre una variable escalar y una tabla. Tenemos dos sintaxis:

Por ejemplo, int Edades[] y int [] Edades son formas equivalentes para definir un vector. En este punto habremos declarado la tabla, pero no la hemos creado todavía.

## OPERADOR new

Nos servirá para crear la tabla con la longitud adecuada. La sintaxis es la siguiente:



Por ejemplo, Edades = new int[5].

Por tanto, este operador va a construir la tabla donde todos los elementos estarán inicializados en 0 si es un array de tipo numércio o en false si es un array de tipo booleano.

Cualquier otro tipo se inicializará en null.

También debemos tener en cuenta que se puede declarar y crear la tabla en una única sentencia:

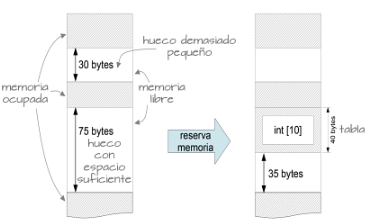
Texto

Descripción generada automáticamente con confianza media

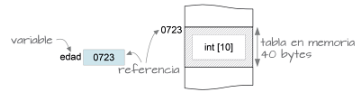
Existe una alternativa sintáctica que sólo será posible si conocemos el valor de todos los elementos que van a formar la tabla. En este caso declararemos la tabla y luego entre llaves introduciremos los diferentes valores que va a tener la tabla.



# REFERENCIAS

Cada dato de un array se va a almacenar en memoria. Una vez que hayamos declarado y creado la tabla, se reservará el espacio en memoria necesario para completarlo, para ello multiplicará la cantidad de datos a guardar por el tipo de dato que se utiliza, por ejemplo, un int ocupa 4 bytes de memoria por lo que un array de 10 datos de tipo int ocupará 40 bytes de memoria.

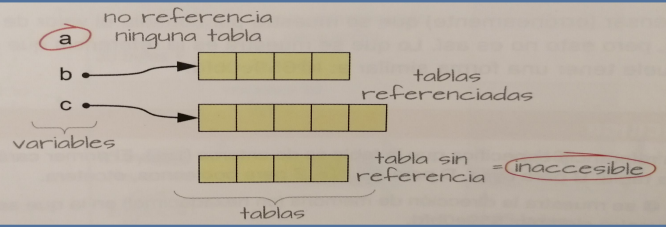
Una vez que se vayan introduciendo los valores, esto se guardarán en cada dirección de memoria asignada. Cada una de esas direcciones de memoria es lo que se denomina referencia.

Por tanto, lo que se va a guardar realmente dentro del array no es el valor que se introduzca por consola sino la referencia (la dirección de memoria donde se está guardando la variable). Por eso también se les conoce como variables de referencia.

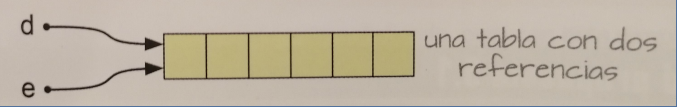
Es decir, si hacemos un array de Edades con 4 índices con los valores 0, 1, 2 y 3 y solicitamos una impresión por consola de Edad[1], en consola se verá el 1 que es el dato que se ha guardado en memoria pero en realidad lo que se estaría imprimiendo es la referencia.

Estas referencias se modifican en cada ejecución, dependiendo de la ocupación de la memoria.

Si tenemos 3 variables array declaradas (por ejemplo, int a[], b[], c[]) e inicializamos b (b = new int [4]) y c (c = new int [5]) y new int [3]. La variable int a[] está declarada pero no hay ninguna referencia a ese array por lo que no hay posibilidad de almacenar datos, sin embargo b y c sí que son útiles.



Otro escenario posible es aquel en el que a través de dos variables diferentes accedamos a la misma tabla, por ejemplo declaramos dos variables int d[] e int e[] e inicializamos ambas de la siguiente manera: d = new int [6] y e = d. De esta manera las referencias de int d[] se guardarán también en int e[]. Esto no quiere decir que se copien los datos del array sino que se accede a las mismas posiciones de memoria desde dos variables diferentes.



Esto sólo es posible hacerlo si los tipos de e y d son iguales, si fueran diferentes (por ejemplo, d un tipo int y e un tipo boolean) habría incompatibilidad de tipos y el programa provocaría un error.

## RECOLECTOR DE BASURA

Java incorpora un sistema para impedir que las variables array declaradas, pero no iniciadas sean borradas y no ocupen un espacio en memoria de forma inútil llegando a dejar grandes cantidades de memoria ocupadas e inaccesibles (bien por un mal diseño, bien por una mala intención).

Este sistema es el recolector de basura, Java de forma periódica comprueba que todas las tablas declaradas están construidas, si encuentra alguna que está declarada pero no construida la destruye dejando libre el espacio de memoria que estaba ocupando hasta ese momento.

## REFERENCIA NULL

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza bajaEstá variable de referencia nos va a servir para dejar una referencia vacía.

# USO DE TABLAS

Podemos trabajar con tablas de dos formas distintas teniendo en cuenta que:

* Sólo algunos elementos de la tabla almacenan información útil.
* Todos los elementos de la tabla almacenan información útil.

Hacer lo segundo va a simplificar la resolución de problemas en Java, por ejemplo, dada la siguiente tabla:

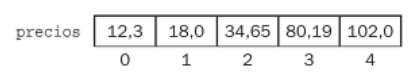


Todos los índices del array tienen un valor de referencia de tipo double que guarda una estatura.

En este caso una vez creadas las tablas, estas mantienen su longitud (de 0 a 4 en este caso) de forma constante y no es posible cambiarla. En el caso de que necesitemos aumentar el tamaño habrá que crear una segunda tabla que tenga una longitud mayor y, en ella, copiar los datos de la tabla original que nos interesen.

## TABLAS ORDENADAS

Son tablas en las que los datos están ordenados en sentido creciente o decreciente, por ejemplo:



En estos casos deberemos tener cuidado con cómo insertar o eliminar datos de la tabla para que esta continue de forma 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.

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.

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

Las tablas también se pueden pasar 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.

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.

# OPERACIONES CON TABLAS: LA CLASE ARRAYS

La API de Java contiene una clase que facilita el trabajo con arrays se trata de la clase Arrays que se integra dentro del paquete java.util.

## OBTENCIÓN DEL NÚMERO DE ELEMENTOS DE UNA TABLA

Para conocer el número de elementos de un array utilizaremos el método nombreArray.length.

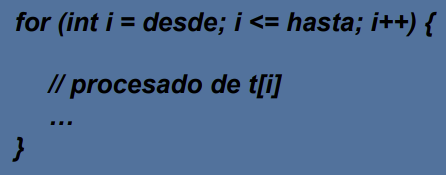
## INICIALIZACIÓN

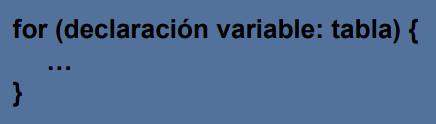
Utilizaremos la inicialización cuando queramos iniciar las variables del array con un valor diferente al que tienen por defecto.

Para ello utilizaremos el método fill, el cual es un método sobrecargado que nos permitirá varias opciones:

* + Arrays.fill(nombreArray, valorDato). Por ejemplo, partiendo de un array de alturas de tipo double vamos a hacer que todas las edades estén inicializadas por defecto en 1.50m: Arrays.fill(rlnAltura, 1.50).
  + Arrays.fill(nombreArray, indiceInicio, indiceFin, valorDato). Utilizando el mismo dato vamos a hacer que los índices entre el 15 y el 20 se inicien en 1.50, el resto lo harán en 0.0: Arrays.fill(rlnAltura, 15, 21, 1.50).

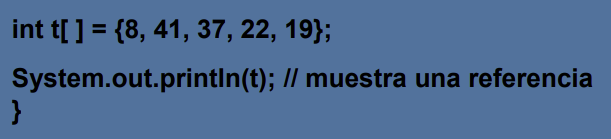
## RECORRIDO

El patrón de código para recorrer una tabla es:

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

## MOSTRAR UNA TABLA

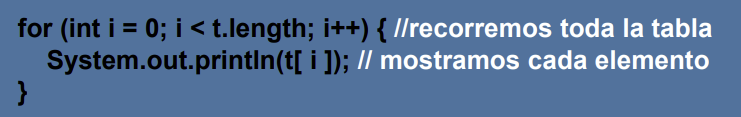
Mostrar una tabla consiste en mostrar sus elementos. Si ejecutamos:



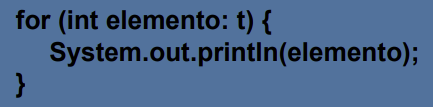
No se muestra el contenido de la tabla; en su lugar se muestra la referencia que contiene t.

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( ).

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



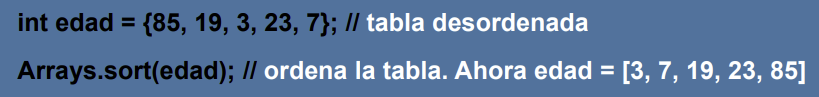
O utilizando for each:



## ORDENACIÓN

Ordenar una tabla consiste en cambiar de posición los datos que contiene para que, en conjunto, resulten ordenados.

Para ordenar una tabla haremos uso del método sort de la clase Arrays.



## BÚSQUEDA

1. BÚSQUEDA EN UNA TABLA NO ORDENADA

Se realizará una búsqueda secuencial recorriendo la tabla y comprobando los valores de los elementos. Esta búsqueda terminara cuando encontremos el valor o cuando hayamos recorrido todo el array.

1. BÚSQUEDA EN UNA TABLA ORDENADA

Se utiliza una búsqueda dicotómica o búsqueda binaria utilizando el método de la clase Arrays binarySearch. Este método busca el valor que queremos dentro de un array ordenado y devuelve un valor positivo o 0 (si encuentra el elemento) que sería el índice del array en el que se encuentra ese valor o un valor negativo (si no encuentra el elemento) que informa de la posición donde tendría que colocarse el elemento buscado para que la tabla continúe ordenada.

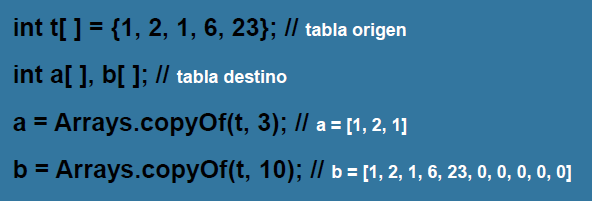
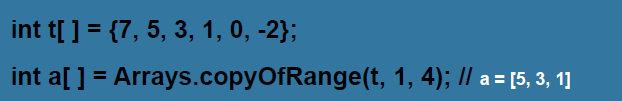
## 

## 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.

Sin embargo, Arrays proporciona esta funcionalidad mediante:

* + Arrays.copyOf(nombreArray, nuevaLongitud). Construye un nuevo array de una nueva longitud.
  + Arrays.copyOfRange(nombreArray, inicio, fin). Construye un nuevo array que copia los valores desde inicio (incluido) hasta fin (excluido).

## INSERCIÓN

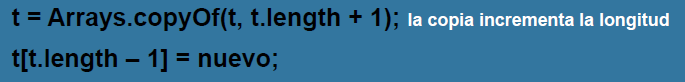
La forma de añadir un nuevo valor a una tabla depende de si está o no ordenada.

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.

1. 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.

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

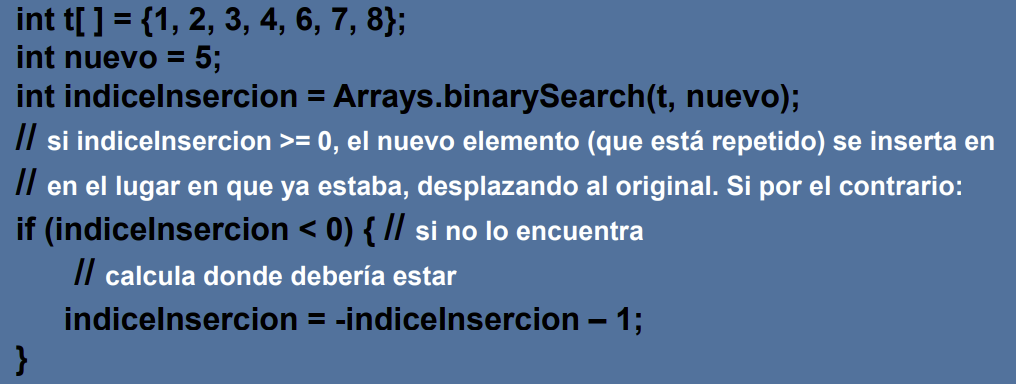
1. INSERCIÓN ORDENADA

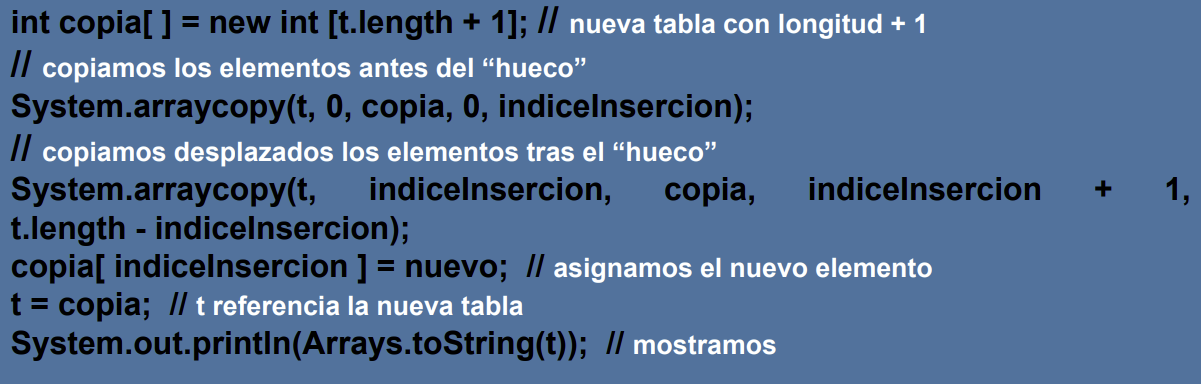
La inserción ordenada consiste en añadir un nuevo elemento en la tabla en la posición adecuada para que la tabla continúe ordenada.

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.

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.

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:



## ELIMINACIÓN

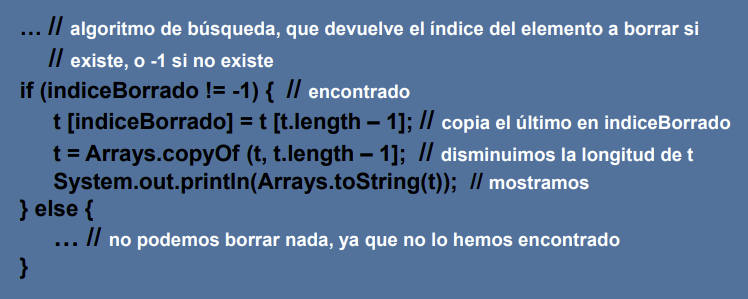
Esta operación consiste en borrar un elemento de la tabla, por lo que después de una eliminación la longitud de la tabla decrece.

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.

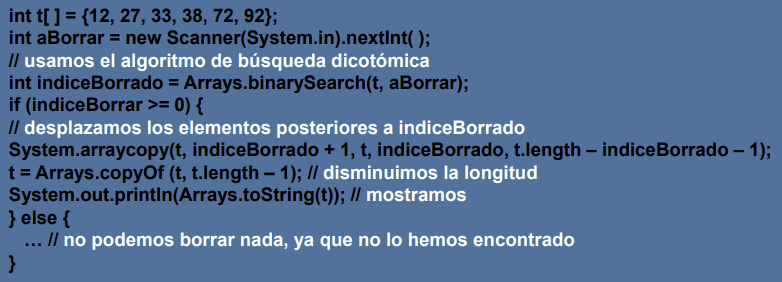
1. TABLA NO ORDENADA

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.

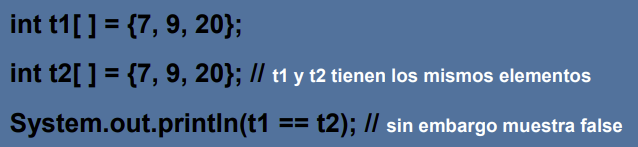
Veamos el algoritmo donde t[ ] es la tabla con los datos e indiceBorrado contendrá, si existe, el índice del elemento que deseamos eliminar, que se almacena en la variable aBorrar.

1. TABLA ORDENADA

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 indiceBorrrado) 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.

## COMPRACIÓN 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:

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

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.



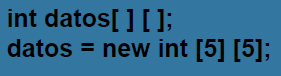
# TABLAS n-DIMENSIONALES

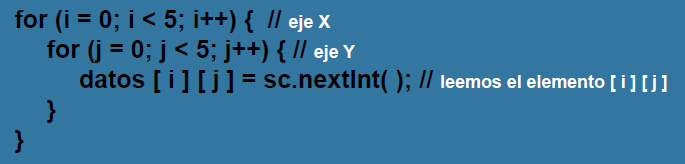
## TABLAS BIDIMENSIONALES

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.

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

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



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:

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.



## 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.

## 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.

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.

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.

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.