CFGS APLICACIONES MULTIPLATAFORMA - PROGRAMACIÓN

UNIDAD 4

ESTRUCTURAS DE ALMACENAMIENTO

DIEGO VALERO ARÉVALO
BASADO EN APUNTES Y EJERCICIOS PROPORCIONADOS POR
Mª CARMEN DÍAZ GONZÁLEZ - IES VIRGEN DE LA PALOMA

U4 - **ESTRUCTURAS DE ALMACENAMIENTO**
ÍNDICE

4.1 - INTRODUCCIÓN A ARRAYS

Los **arrays** o **tablas** son **estructuras de almacenamiento** que podríamos resumir como **"variables que contienen múltiples valores"**.

Son un tipo de colección en el cual **se almacenan datos del mismo tipo** que se colocan en **posiciones numeradas**. Son útiles para **agrupar** variables en vez de declararlas todas por separado.

Vamos a ver dos tipos de arrays:

Unidimensionales	Multidimensionales o matrices

>> 4.1.1- FUNCIONAMIENTO BÁSICO DE UN ARRAY

Imagina que tenemos una **mesa** con varios **cajones** en fila que están numerados y empiezan por el **número 0**, y que cada uno de esos cajones sólo puede tener una cosa.

Pero no podemos empezar a meter cosas en los cajones **hasta que no especifiquemos de qué tipo son estas**, para poder saber exactamente qué es lo que vamos a guardar. Por ejemplo, si queremos guardar una manzana, una naranja, una pera,... tendremos que saber que la mesa será sólo para frutas.

4.2 - ARRAYS UNIDIMENSIONALES

>> 4.2.1- DECLARAR UN ARRAY UNIDIMENSIONAL

Si trasladamos el ejemplo de la mesa a programación, lo que queremos hacer es un **array**, el cual se empieza a construir de la siguiente manera:

```
tipo[] nombreArray; / tipo nombreArray[];
```

Para especificar que una variable es un array, se usa `[]`, que se puede poner **detrás del tipo (recomendada)** o **del nombre**, es indiferente.

Ahora, los arrays tienen un **tamaño predefinido** para saber cuántas cosas pueden guardar. Podemos establecerlo de dos maneras:

POR CANTIDAD DE DATOS	POR TAMAÑO n
`tipo[] nombreArray = {a, ..., n};`	`tipo[] nombreArray = new tipo[n];`

Dependiendo de cuántos datos por defecto añadamos al array, este automáticamente registrará esa cantidad como el tamaño máximo.	Asignando esta estructura donde `n` es un número definimos el tamaño directamente.

Ahora que ya sabemos cómo funciona un array, podemos transformar la mesa a un diagrama y crear un ejemplo:

EJEMPLO

"manzana"	"naranja"	"pera"	"kiwi"	"plátano"	"sandía"
0	1	2	3	4	5

El tamaño de la mesa es **6**, y queremos guardar una **fruta** en cada cajón, las cuales queremos que sean de tipo **string**.

Podemos declararlo con el **tamaño** y añadir las frutas después o **directamente** con las frutas que queramos:

.java

```java
String[] frutas = new String[6];
```

```java
String[] frutas = {"manzana","naranja","pera","kiwi","plátano","sandía"};
```

>> 4.2.2- AÑADIR VALORES A UN ARRAY UNIDIMENSIONAL

Si sabemos el tamaño de nuestro array y el tipo de datos, pero aún **no sabemos qué vamos a guardar**, declararemos entonces el array por su tamaño, pero ¿cómo guardamos las cosas después? Tendremos que inicializar el array, y para ello hay que acceder a sus **posiciones**.

```java
nombreArray[x] = valor;
```

`nombreArray[]`	Para acceder a una posición del array, escribimos su nombre junto a `[]`.
`x`	Dentro de `[]` escribimos el **número de la posición (`x`)** donde queremos guardar un `valor`. ⚠ El número de la posición **nunca puede ser mayor o igual al tamaño del array**, porque no existe en el array y saltaría una excepción `OutOfBounds` **(fuera de los límites)**.

EJEMPLO

Persona.java

```java
frutas[1] = "naranja";
```

Aquí hemos añadido al **segundo elemento** (cuyo **número de posición** es `1`) del array **frutas**, el valor "naranja".

💡 La **última posición** de un array siempre será **su tamaño - 1**. Para **saber el tamaño de un array**, este trae consigo un método que devuelve ese número: `.length`.

EJEMPLO

.java	Console

```java
1  System.out.println(frutas.length);
```
```
6
```

>> 4.2.3- ACCEDER A LOS ELEMENTOS DE UN ARRAY UNIDIMENSIONAL

Para acceder a un elemento hacemos exactamente igual que cuando queremos añadir un valor: llamamos al **array** y al **número de la posición** que nos interesa.

```java
nombreArray[x];
```

EJEMPLO

.java
```java
1  System.out.println(frutas[1]);
```
Console
```
naranja
```

>> 4.2.4- RECORRER UN ARRAY UNIDIMENSIONAL

Es muy común que necesitemos recorrer todos los elementos de un array para, por ejemplo, comprobar si un dato existe dentro del array o simplemente si queremos imprimir por consola cada elemento. Para ello podemos usar un bucle **`for`** que comience en la primera posición (`0`) y llegue hasta **antes de alcanzar el tamaño** o hasta **su tamaño-1**:

```java
for(int i = 0; i < nombreArray.length; / i <= nombreArray.length-1; i++){
  ...
}
```

EJEMPLO

.java
```java
1  for(int i = 0; i < frutas.length; i++){
2     System.out.println(frutas[i]+", ");
3  }
```
Console
```
manzana, naranja, pera, kiwi,
plátano, sandía,
```

>> 4.2.5- BÚSQUEDAS Y ORDENACIONES DE UN ARRAY UNIDIMENSIONAL

Buscar en un array tiene mucho que ver con **ordenarlo**, ya que en un array no ordenado solo hay una manera de buscar un elemento: recorrer el array comprobando uno a uno.

JAVA tiene predefinidos en la clase **`Arrays`** varios métodos que ordenan estos automáticamente.

TEN EN CUENTA

Las búsquedas y ordenaciones son operaciones que necesitan muchos recursos, y cuanto más grande sea el array que vamos a utilizar, más tiempo y memoria requerirá. **Estas operaciones no son recomendadas para arrays grandes.**

· 4.2.5.1- BÚSQUEDAS

Tenemos **dos métodos de búsqueda** principales:

Secuencial	Dicotómica

BÚSQUEDA SECUENCIAL

La **búsqueda secuencial** es la más fácil de las dos ya que consiste en **comparar los elementos del vector (`nombreArray[i]`) con el elemento a buscar**.

```
for(int i = 0; i < nombreArray.length; i++){
   if(nombreArray[i] == elemento){
      ...
   }
}
```

Un ejemplo de uso de este método es **almacenar la posición del vector** cuando este coincida con el elemento.

EJEMPLO

.java

```
1  int[] arrayNumeros = {4,3,2,5,1};
2  int elemento = 5;
3  int posicion;
4
5  for(int i = 0; i < arrayNumeros.length; i++){
6     if(arrayNumeros[i] == elemento){
7        posicion = arrayNumeros[i];
8     }
9  }
10
11 System.out.println("La posición del elemento es "+posicion);
```

Console

```
La posición del elemento es 3
```

BÚSQUEDA DICOTÓMICA

La **búsqueda dicotómica** divide un array en dos y lo recorre **desde la izquierda** o **desde la derecha hacia el centro** hasta **encontrar** el elemento. Esto se define calculando el valor más cercano al centro y comparándolo con el elemento.

```
int[] array = ...;
int elemento = n;
int izda = 0;
int dcha = array.length-1;
int centro = (izda+dcha)/2;
int posicion = -1;

while (izda<=dcha && array[centro] != elemento){
   if(n<array[cent]) {
      dcha = centro-1;
   }
   else {
      izda = centro+1;
   }
   cent = (izda+dcha)/2;
}

if(izda>dcha) {
   posicion = -1;
}
else {
   posicion = centro;
}
```

La única **restricción** para usar la búsqueda dicotómica es **que el array tiene que estar ya ordenado**.

Un ejemplo de uso de este método es **almacenar la posición del vector** cuando este coincida con el

elemento.

EJEMPLO

.java

```java
int[] arrayNumeros = {1,2,3,4,5,6,7,8,9,10};
int elemento = 2;
int izda = 0;
int dcha = array.length-1;
int centro = (izda+dcha)/2;
int posicion = -1;

while (izda<=dcha && array[centro] != elemento){
   if(n<array[cent]) {
      dcha = centro-1;
   }
   else {
      izda = centro+1;
   }
   cent = (izda+dcha)/2;
}

if(izda>dcha) {
   posicion = -1;
}
else {
   posicion = centro;
}

if(posicion = = -1) {
   System.out.println("El elemento no está en el array");
}
else {
   System.out.println("La posición del elemento es "+posicion);
}
```

Console

```
La posición del
elemento es 1
```

· 4.2.5.2- ORDENACIONES

Existen **cuatro métodos de ordenación** principales:

Bubble	Insertion	Selection	Quicksort

Aunque son interesantes de conocer, no es necesario saber cómo funcionan a fondo ya que JAVA los trae implementados por defecto. Las ordenaciones de **`arrays`** pueden hacerse con el método **`.sort`** que trae la propia clase **`Arrays`**.

```java
Arrays.sort(nombreArray);
```

EJEMPLO

.java

```java
int[] arrayNumeros = {8,7,9,6,4,5,2,3,1,10};

for(int i = 0; i < arrayNumeros.length; i++){
   System.out.println("Antes de ordenar:");
   System.out.println(arrayNumeros[i]+", ");
}

Arrays.sort(arrayNumeros);
System.out.println();

for(int i = 0; i < arrayNumeros.length; i++){
   System.out.println("Después de ordenar:");
   System.out.println(arrayNumeros[i]+", ");
}
```

Console

```
Antes de ordenar:
8, 7, 9, 6, 4, 5, 2, 3, 1, 10,

Después de ordenar:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

Vamos a ver la clase **`Arrays`** más a fondo el siguiente apartado.

>> 4.2.6- MÉTODOS DE LA CLASE `ARRAYS`

Vamos a ver algunos de los **métodos más utilizados** dentro de esta clase que nos permite manipular **`arrays`** más fácilmente:

`.fill(array, valor);`	Rellena un **`array`** con el mismo **`valor`** en todas las posiciones.
`.equals(arrayA, arrayB);`	Compara **`arrayA`** con **`arrayB`** y devuelve un **booleano**. Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores, da igual cómo estén colocados.

`.sort(array);`
 	
`.sort(array, posA, posB);`
	Como hemos visto en el apartado anterior, ordena los elementos de un **`array`** en orden ascendente utilizando el alfabeto ASCII.
	
	También podemos ordenar **sólo una parte del array**, especificando las posiciones desde la **`posA`** hasta la **`posB`**.

`.binarySearch(array, elem);`	Permite buscar un **`elemento`** de forma ultrarrápida en un **`array`** **ordenado**.

4.3 - ARRAYS MULTIDIMENSIONALES O MATRICES

>> 4.2.1- DECLARAR UN ARRAY MULTIDIMENSIONAL O MATRIZ

Podemos almacenar **arrays dentro de arrays**, esto es lo que se conoce como **arrays multidimensionales o matrices**. Para crear uno lo definimos de la siguiente manera:

```
tipo[][] nombreMatriz ; / tipo nombreMatriz [][];
```

Al igual que con los array, se usa **`[][]`**, que se puede poner **detrás del tipo (recomendada)** o **del nombre**, es indiferente.

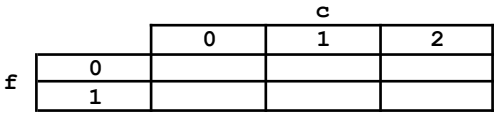
No podemos rellenar una matriz directamente, así que lo siguiente es definir el tamaño de las **filas** y de las **columnas**:

```
tipo[][] nombreMatriz = new tipo[f][c];
```

EJEMPLO

.java

```
1  int[][] miMatriz = new int[2][3];
```



>> 4.2.2- AÑADIR VALORES A UNA MATRIZ

Para acceder a las posiciones de una matriz debemos **especificar** el número de la **fila** y la **columna**.

```
nombreMatriz[f][c] = valor;
```

`nombreMatriz[][]`	Para acceder a una posición del array, escribimos su nombre junto a `[][]`.
`[f][c]`	Dentro de `[][]` escribimos el **número de la fila (`f`)** y **de la columna (`c`)** donde queremos guardar un **`valor`**. ⚠ El número de las posiciones **nunca puede ser mayor o igual al tamaño de las filas** o **de las columnas** respectivamente, porque no existen en la matriz y saltaría una excepción **`OutOfBounds` (fuera de los límites)**.

EJEMPLO

.java

```
1  miMatriz[0][1] = 7.5;
2  miMatriz[1][1] = 12;
3  miMatriz[2][0] = 3.63;
```

f \ c	0	1	2
0		7.5	
1	3.63	12	

>> 4.2.3- ACCEDER A LOS ELEMENTOS DE UNA MATRIZ

Para acceder a un elemento hacemos exactamente igual que cuando queremos añadir un valor: llamamos a la **matriz** y al **número de la fila y de la columna** que nos interesa.

```
nombreMatriz[f][c];
```

EJEMPLO

.java

```
1  System.out.println(miMatriz[0][2]);
```

Console

```
null
```

>> 4.2.4- RECORRER UNA MATRIZ

La manera de recorrer una matriz es a través de **dos bucles `for` anidados**: el primero recorrerá las **filas** y el segundo las **columnas de esa fila**.

```
for(int fila = 0; fila < nombreMatriz.length; fila++){
   for(int columna = 0; columna < nombreMatriz[fila].length; columna++){
      ...
   }
}
```

El **`.length`** por defecto de una matriz devuelve el **número de filas**, y si accedemos a **cualquiera de las filas** podremos saber **el número de columnas totales**, porque todas las filas son igual de largas.

Esto es muy útil para rellenar las posiciones de una matriz con valores.

4.4 - CLASE STRING

La clase **`String`** nos permite trabajar con conjuntos de caracteres complejos. Ya sabes que en esencia un **`String`** es un **`array`** de **`char`**.

Puedes **concatenar** varias **líneas de `strings`** o **`strings` y otras variables** con el operador **+**.

EJEMPLO

.java

```
1  String texto = "Este es un texto que ocupa " +
2                 "varias líneas, pero lo podemos " +
3                 "concatenar perfectamente";
```

>> 4.4.1- COMPARACIÓN (MÉTODOS `EQUALS` Y `COMPARE TO`)

Como los objetos String son diferentes a otros tipos de variables, **no podemos usar el operador == para compararlos**. En su lugar tenemos los siguientes métodos:

`stringA.equals(stringB);`	El resultado es true si **`stringA`** es igual a **`stringB`**. Ambas son variables de tipo **`String`**.
`stringA.equalsIgnoreCase(stringB);`	Igual que la anterior, pero ignora las mayúsculas y minúsculas.
`stringA.compareTo(stringB);`	Compara ambas cadenas, considerando el **orden alfabético ASCII** (donde la letra **ñ y las tildes** son mayores que otras letras). Si **`stringA`** es **mayor en orden alfabético** que **`stringB`**, devuelve **un número mayor que** `0`, si son **iguales devuelve** `0` y si **es la segunda la mayor, devuelve un número menor que** `0`.
`stringA.compareToIgnoreCase(stringB);`	Igual que la anterior, pero ignora las mayúsculas y minúsculas.

>> 4.4.2- MÉTODOS DE LA CLASE `STRING`

Vamos a ver algunos de los **métodos más utilizados** dentro de esta clase que nos permite manipular **`strings`** más fácilmente:

`.valueOf(var);`	Convierte el **tipo** de **`var`** a **`String`**.
`string.length(string);`	Devuelve la longitud del **`string`**.
`stringA + stringB;` `stringA.concat(stringB);`	Para **concatenar** cadenas. Disponemos del operador **+** o del método **`.concat`**.
`string.charAt(n);`	Devuelve el **`char`** que se encuentra en la **posición `n`** del **`string`**.

`string.substring(posA, posB);`	Devuelve la porción de **`string`** desde **`posA`** **hasta la anterior a** **`posB`**. Si las posiciones **no existen** en el **`string`** saltaría una excepción **`OutOfBounds`** **(fuera de los límites)**.

`stringA.indexOf(stringB);`
 	
`stringA.indexOf(stringB, pos);`
	Devuelve la **primera posición** en la que aparece **`stringB`** en **`stringA`**. En el caso de que **`stringB`** no se encuentre, devolverá **`-1`**. **`stringB`** puede ser **`char`** o **`String`**.
	
	También podemos especificar a partir de qué **posicion** (**`pos`**) queremos que busque.

`stringA.lastIndexOf(stringB);`
 	
`stringA.lastIndexOf(stringB, pos);`
	Devuelve la **última posición** en la que aparece **`stringB`** en **`stringA`**, ignorando los anteriores si se repiten.
	
	También podemos especificar a partir de qué **posicion** (**`pos`**) queremos que busque.

`stringA.endsWith(stringB);`
 	
`stringA.startsWith(stringB);`
	Estos métodos devolverán **true** si **`stringA`** termina (**ends**) o empieza (**starts**) con **`stringB`** .

`string.replace(stringA, stringB);`
 	
`string.replaceFirst(stringA, stringB);`
 	
`string.replaceAll(stringA, regex);`
 	
Estos métodos son muy útiles para quitar algún carácter en específico. La sustitución puede incluso hacerse con un string vacío (`""`).
	Cambia **todas las apariciones de un `char` o `String` en `stringA`** por **lo que se especifique en `stringB`** y lo almacena como resultado. **El texto original no se cambia**, por lo que hay que asignar el resultado a un nuevo **`String`** para conservar el texto cambiado.
	
	`.replaceFirst` sólo afecta a la primera coincidencia, y **`.replaceAll`** se utiliza si en vez de otro **`string`** queremos usar **expresiones regulares**.

`string.toUpperCase();`
 	
`string.toLowerCase();`
	Convierte **todos los caracteres de `string`** a **mayúsculas** (**`Upper`**) o a **minúsculas** (**`Lower`**).

`string.toCharArray();`	Convierte un **`string`** a un **`array`** de tipo **`char`**. Esto nos permite **manipular un `string` de forma más fácil** usando los métodos de la clase **`Arrays`**.
`.format("%.nf", var);`	Modifica el **formato del `string` a mostrar**. Muy útil para mostrar **sólo los decimales que necesitemos de un número decimal**. Indicaremos **`%`** para indicar **la parte entera** más **el número de decimales a mostrar (`n`) seguido de `f`**.
`string.split(regex);`	Devuelve un **`array`** de tipo **`String`**, el cual almacenará en cada posición un **fragmento del `string`**, que se divide en cada parte que coincida con ***`regex`***, que puede ser una expresión regular, una letra, un signo,... Esta parte **no aparece en el `array` final**.
`string.trim();`	Devuelve una copia de la cadena **eliminando los espacios en blanco de ambos extremos de `string`**. No afecta a aquellos que estén en medio.

U4 - **BATERÍA DE EJERCICIOS**

1	Crea una biblioteca de funciones (librería) para arrays (de una dimensión) de números enteros que contenga las siguientes funciones: a) generaArrayInt: Genera un array de tamaño n con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro. b) minimoArrayInt: Devuelve el mínimo del array que se pasa como parámetro. c) maximoArrayInt: Devuelve el máximo del array que se pasa como parámetro. d) mediaArrayInt: Devuelve la media del array que se pasa como parámetro. e) estaEnArrayInt: Dice si un número está o no dentro de un array. f) posicionEnArray: Busca un número en un array y devuelve la posición (el índice) en la que se encuentra. g) volteaArrayInt: Le da la vuelta a un array. Crea una clase para probar todas las funciones de la librería, además está podrá ser utilizada en cualquier otro ejercicio.
2	Crea un programa para realizar cálculos relacionados con la altura (en metros) de personas. Pedirá un valor N y luego almacenará en un array N alturas introducidas por teclado. Luego mostrará la altura media, máxima y mínima así como cuántas personas miden por encima y por debajo de la media.
3	Crea un programa que cree un array de 10 enteros y luego muestre el siguiente menú con distintas opciones: a) Mostrar valores. b) Introducir valor. c) Salir. La opción 'a' mostrará todos los valores por pantalla. La opción 'b' pedirá un valor V y una posición P, luego escribirá V en la posición P del array. El menú se repetirá indefinidamente hasta que el usuario elija la opción 'c' que terminará el programa.
4	Crea un programa que permita al usuario almacenar una secuencia aritmética en un array y luego mostrarla. Una secuencia aritmética es una serie de números que comienza por un valor inicial V, y continúa con incrementos de I. Por ejemplo, con V=1 e I=2, la secuencia sería 1, 3, 5, 7, 9... Con V=7 e I=10, la secuencia sería 7, 17, 27, 37... El programa solicitará al usuario V, I además de N (no de valores a crear).

NOTA : Utiliza los métodos de la clase 'Arrays' para ayudarte a resolver los siguientes ejercicios:

5	Crea un programa que pida al usuario dos valores N y M y luego cree un array de tamaño N que contenga M en todas sus posiciones. Luego muestra el array por pantalla.
6	Crea un programa que cree un array de números enteros e introduzca la siguiente secuencia de valores: `1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5,` etc. hasta introducir 10 diez veces, y luego la muestre por pantalla. En esta ocasión has de utilizar Arrays.fill().

7	Crea un programa que pida al usuario 20 valores enteros e introduzca los 10 primeros en un array y los 10 últimos en otro array. Por último, comparará ambos arrays y le dirá al usuario si son iguales o no.
8	Crea un programa que cree un array de tamaño 30 y lo rellene con valores aleatorios entre 0 y 9 . Luego ordena los valores del array y los mostrará por pantalla.
9	Necesitamos crear un programa para mostrar el ranking de puntuaciones de un torneo de ajedrez con 8 jugadores. Se le pedirá al usuario que introduzca las puntuaciones de todos los jugadores (habitualmente valores entre 1000 y 2800, de tipo entero) y luego muestre las puntuaciones en orden descendente (de la más alta a la más baja).
10	. Crea un programa que cree un array de tamaño 1000 y lo rellene con valores enteros aleatorios entre 0 y 99. Luego pedirá por teclado un valor N y se mostrará por pantalla si N existe en el array, además de cuantas veces.
11	Define tres arrays de 20 números enteros cada una, con nombres numero, cuadrado y cubo. Carga el array numero con valores aleatorios entre 0 y 100. En el array cuadrado se deben almacenar los cuadrados de los valores que hay en el array numero. En el array cubo se deben almacenar los cubos de los valores que hay en numero. A continuación, muestra el contenido de los tres arrays dispuesto en tres columnas.
12	Escribe un programa que lea 15 números por teclado y que los almacene en un array. Rota los elementos de ese array, es decir, el elemento de la posición 0 debe pasar a la posición 1, el de la 1 a la 2, etc. El número que se encuentra en la última posición debe pasar a la posición 0. Finalmente, muestra el contenido del array.
13	Escribe un programa que genere 100 números aleatorios del 0 al 20 y que los muestre por pantalla separados por espacios. El programa pedirá entonces por teclado dos valores y a continuación cambiará todas las ocurrencias del primer valor por el segundo en la lista generada anteriormente. Los números que se han cambiado deben aparecer entrecomillados.
14	Escribe un programa que rellene un array de 20 elementos con números enteros aleatorios comprendidos entre 0 y 400 (ambos incluidos). A continuación el programa mostrará el array y preguntará si el usuario quiere resaltar los múltiplos de 5 o los múltiplos de 7. Seguidamente se volverá a mostrar el array escribiendo los números que se quieren resaltar entre corchetes.
15	Un restaurante nos ha encargado una aplicación para colocar a los clientes en sus mesas. En una mesa se pueden sentar de 0 (mesa vacía) a 4 comensales (mesa llena). Cuando llega un cliente se le pregunta cuántos son. De momento el programa no está preparado para colocar a grupos mayores a 4, por tanto, si un cliente dice por ejemplo que son un grupo de 6, el programa dará el mensaje “Lo siento, no admitimos grupos de 6, haga grupos de 4 personas como máximo e intente de nuevo”. Para el grupo que llega, se busca siempre la primera mesa libre (con 0 personas). Si no quedan mesas libres, se busca donde haya un hueco para todo el grupo, por ejemplo si el grupo es de dos personas, se podrá colocar donde haya una o dos personas. Inicialmente, las mesas se cargan con valores aleatorios entre 0 y 4. Cada vez que se sientan nuevos clientes se debe mostrar el estado de las mesas. Los grupos no se pueden romper aunque haya huecos sueltos suficientes. El funcionamiento del programa se muestra a continuación:

```
|Mesa num:   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
|Ocupación: | 1 | 0 | 2 | 4 | 2 | 2 | 3 | 4 | 0 | 2

¿Cuántos son? (Introduzca -1 para salir del programa): 4

Por favor, siéntense en la mesa número 2.

|Mesa num:   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
|Ocupación: | 1 | 4 | 2 | 4 | 2 | 2 | 3 | 4 | 0 | 2

¿Cuántos son? (Introduzca -1 para salir del programa): 3

Por favor, siéntense en la mesa número 9.

|Mesa num:   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
|Ocupación: | 1 | 4 | 2 | 4 | 2 | 2 | 3 | 4 | 3 | 2

¿Cuántos son? (Introduzca -1 para salir del programa): 7

Lo siento, no admitimos grupos de 7, haga grupos de 4 personas como máximo e intente de nuevo.
|Mesa num:   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
|Ocupación: | 1 | 4 | 2 | 4 | 2 | 2 | 3 | 4 | 3 | 2

¿Cuántos son? (Introduzca -1 para salir del programa): 3

Tendrán que compartir mesa. Por favor, siéntense en la mesa número 1.

|Mesa num:   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
|Ocupación: | 4 | 4 | 2 | 4 | 2 | 2 | 3 | 4 | 3 | 2

¿Cuántos son? (Introduzca -1 para salir del programa): -1
Gracias por su visita.
```

16 Crea un programa que cree una matriz de tamaño 5x5 que almacene los números del 1 al 25 y luego muestre la matriz por pantalla.

17 Define un array de números enteros de 3 filas por 6 columnas con nombre num y asigna los valores según la siguiente tabla. Muestra el contenido de todos los elementos del array dispuestos en forma de tabla como se muestra a continuación.

	Columna 0	Columna 1	Columna 2	Columna 3	Columna 4	Columna 5
Fila 0	0	30	2	0	0	7
Fila 1	75	0	0	0	0	0
Fila 2	0	0	-2	9	0	11

18 Crea un programa que cree una matriz de 10x10 e introduzca los valores de las tablas de multiplicar del 1 al 10 (cada tabla en una fila). Luego mostrará la matriz por pantalla.

19 Crea un programa que cree una matriz de tamaño NxM (tamaño introducido por teclado) e introduzca en ella NxM valores (también introducidos por teclado). Luego deberá recorrer la matriz y al final mostrar por pantalla cuántos valores son mayores que cero, cuántos son menores que cero y cuántos son igual a cero.

20 Escribe un programa que pida 20 números enteros. Estos números se deben introducir en un array de 4 filas por 5 columnas. El programa mostrará las sumas parciales de filas y columnas igual que si de una hoja de cálculo se tratara. La suma total debe aparecer en la esquina inferior derecha.

					Σ fila 0
					Σ fila 1
					Σ fila 2
					Σ fila 3
Σ columna 0	Σ columna 1	Σ columna 2	Σ columna 3	Σ columna 4	TOTAL

21 Realiza un programa que rellene un array de 6 filas por 10 columnas con números enteros positivos comprendidos entre 0 y 1000 (ambos incluidos). A continuación, el programa deberá dar la posición tanto del máximo como del mínimo.

22 Necesitamos crear un programa para almacenar las notas de 4 alumnos (llamados “Alumno 1”, “Alumno 2”, etc.) y 5 asignaturas. El usuario introducirá las notas por teclado y luego el programa mostrará la nota mínima, máxima y media de cada alumno. (no es necesario guardar el nombre del alumno, sabremos que alumno es por la fila) .

23	Necesitamos crear un programa para registrar sueldos de hombres y mujeres de una empresa y detectar si existe brecha salarial entre ambos. El programa pedirá por teclado la información de N personas distintas (valor también introducido por teclado). Para cada persona, pedirá su género (0 para varón y 1 para mujer) y su sueldo. Esta información debe guardarse en una única matriz. Luego se mostrará por pantalla el sueldo medio de cada género.
24	Crea un programa que pida una cadena de texto por teclado y luego muestre cada palabra de la cadena en una línea distinta.
25	Crea un programa que pida dos cadenas de texto por teclado y luego indique si son iguales, además de si son iguales sin diferenciar entre mayúsculas y minúsculas.
26	Crea un programa que pida por teclado tres cadenas de texto: nombre y dos apellidos. Luego mostrará un código de usuario (en mayúsculas) formado por la concatenación de las tres primeras letras de cada uno de ellos. Por ejemplo si se introduce “Juan”, “Moreno” y “Carmona” mostrará “JUAMORCAR”.
27	Crea un programa que muestre por pantalla cuantas vocales de cada tipo hay (cuantas ‘a’, cuantas ‘e’, etc.) en una frase introducida por teclado. No se debe diferenciar entre mayúsculas y minúsculas. Por ejemplo dada la frase “Mi mama me mima” dirá que hay: • Nº de A's: 3 • Nº de E's: 1 • Nº de I's: 2 • Nº de O's: 0 • Nº de U's: 0
28	Crea un programa que cuente las vocales y consonantes de una frase. Ten cuidado con los espacios, se deben ignorar.
29	Crear un programa que dada una cadena invertir la misma y mostrar por pantalla. Ejemplo: Entrada: "casa blanca" Salida: "acnalb asac"
30	Crear un programa que dada una cadena, y un carácter, indique cuántas veces se repite el carácter en la cadena
31	Crear un programa que lea una frase y encuentre la palabra de mayor longitud. El programa debe imprimir tanto la palabra como el número de caracteres de la misma.
32	Realiza un programa que lea una frase por teclado e indique si la frase es un palíndromo o no (ignorando espacios y sin diferenciar entre mayúsculas y minúsculas). Supondremos que el usuario solo introducirá letras y espacios (ni comas, ni puntos, ni acentos, etc.). Un palíndromo es un texto que se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo: *- Arriba la birra* *- Allí va Ramón y no maravilla* *- Ana lava lana* *- Ella te da detalle*
33	Crear un programa que multiplique dos matrices 4x4.

34	Programa que genera una matriz representando un triángulo invertido alineado a la izquierda realizado con asteriscos, cuyo tamaño será especificado por el usuario. La matriz será mostrada en pantalla finalmente. Por ejemplo, si se indica el tamaño 5 deberá aparecer:
	`*****`
	`****`
	`***`
	`**`
	`*`

35	Ejercicio sala de cine: • Crear una clase enumerada Reserva con los valores libre y ocupado.

• Crear un clase Asiento con las el atributo enumerado y otros que consideres oportunos, una numeración, hora, etc.

• Crear la clase principal e introducir una matriz de 3x2 que se llame sala y poblarla con 6 asientos, todos ellos por defecto a libre.

• Simular con un bucle (do-while) la petición de una fila y una columna del cine y comprobar si esta libre para dejarla ocupada por el espectador hasta que introduzca por ejemplo false y no quiera pedir más entradas o hasta que el cine se complete (este todo ocupado).

• Mostrar para finalizar como se quedo la sesión de esa sala.

Nota: Se puede crear un menú que permita ver la ocupación actual y reservar asiento.