ARRAY DE DOS DIMENSIONES
En java un array de dos dimensiones se consigue con un array unidimensional donde cada elemento almacena una referencia a otro array.

¿Un array de dos dimensiones es un array de arrays?
Si un array de int, en cada uno de sus elementos almacena un int ¿un array de arrays almacena en cada elemento un array?. NO. No es esto posible en java. En java se almacena dentro de un array o bien un tipo primitivo o bien una referencia a un objeto. Lo correcto sería indicar por tanto que un array de dos dimensiones es un array de referencias a otros arrays. No obstante es usual "relajar" la precisión del habla y podemos decir también que un array de dos dimensiones es un *array de arrays*.

Ejemplo:

```
class Unidad4 {

   public static void main(String args[]) {
      int[][] arrayDosD = new int[3][4];
      for (int i = 0; i < 3; i++) {
         for (int j = 0; j < 4; j++) {
            arrayDosD[i][j] = i * j + i + j * 2; //cargamos la matriz con valores sin importancia
            System.out.println("arrayDosD[" + i + "][" + j + "]=" + arrayDosD[i][j]);
         }
      }
   }
}
```

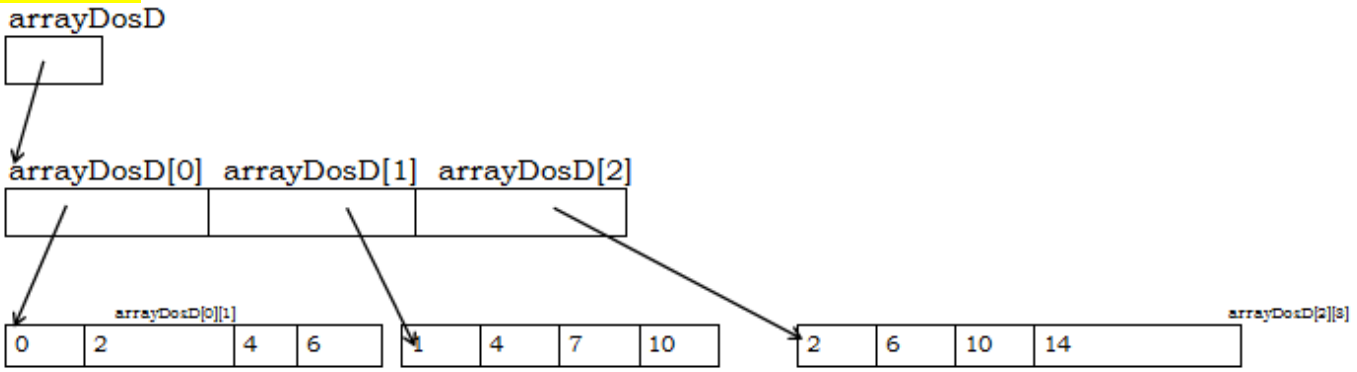
Podemos razonar un array de dos dimensiones como la típica tabla
La visualización gráfica tradicional de un array de dos dimensiones, es el de la típica "tabla" o matriz matemática. Para el ejemplo anterior:

	0	1	2	3
0	**0**	**2**	**4**	**6**
1	**1**	**4**	**7**	**10**
2	**2**	**6**	**10**	**14**

Cuando usamos por ejemplo *arrayDosD[1][2]* "accedemos a la fila 1 columna 2 de la tabla arrayDosD"
Con el código anterior, el bucle externo se encarga de cambiar de fila y el interno de columna.

Como se almacena realmente un array de dos dimensiones.
Los lenguajes de programación, a través del acceso con dos índices, nos permiten razonar un array de dos dimensiones como si fuera una tabla, pero realmente no almacena esa tabla. Imaginar una array de dos dimensiones como una tabla permite razonar cómodamente multitud de problemas lógicos, pero pero además es necesario saber cómo almacena Java ava internamente un array de dos dimensiones por varias razones:
- para más de dos dimensiones esta visión se complica (tres dimensiones es un cubo...)
- NO podemos trabajar con arrays irregulares(es regular cuando es simétrico, es decir, todas las filas tienen las mismas columnas)
- es mejor pensar tal y como trabaja java para aprovechar toda su potencia.

En realidad Java, crea la siguiente estructura en memoria basada en arrays de sólo una dimensión:



- *arrayDosD* es una variable referencia a un array.
- Cada elemento del array anterior es a su vez una referencia a un array de enteros:
  - arrayDosD[0] es una referencia a un array de enteros
  - arrayDosD[1] es una referencia a un array de enteros
  - arrayDosD[2] es una referencia a un array de enteros
- Si tuviéramos una variable x que fuera una referencia a un array de int, podemos asignarle una fila de arrayDosD a x

```
class Unidad4 {

    public static void main(String args[]) {
        int[][] arrayDosD = new int[3][4];
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 4; j++) {
                arrayDosD[i][j] = i * j + i + j * 2; //cargamos la matriz con valores sin importancia
                System.out.println("arrayDosD[" + i + "][" + j + "]=" + arrayDosD[i][j]);
            }
        }
        int[] x;
        x = arrayDosD[0];
        //x[1]  accede al mismo elemento que arrayDosD[0][1]
        System.out.println(x[1]);
        System.out.println(arrayDosD[0][1]);
    }
}
```

Creando un array de dos dimensiones posponiendo la creación de los arrays de segundo nivel.

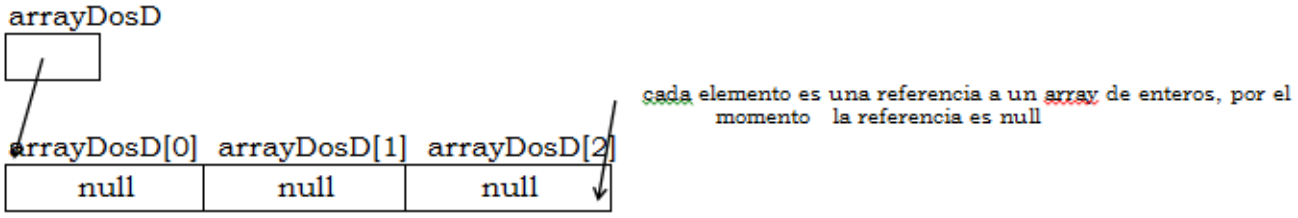
```
int[][] arrayDosD=new int[3][4]
```

Repasa el dibujo de más arriba para recordar que el new anterior nos permite trabajar con un array de dos dimensiones pero que realmente el almacenamiento en memoria de dicho array se simula creando 4 arrays de una dimensión que se organizan en dos niveles, un nivel para cada dimensión.

se puede posponer la creación de los arrays de segundo nivel haciendo:

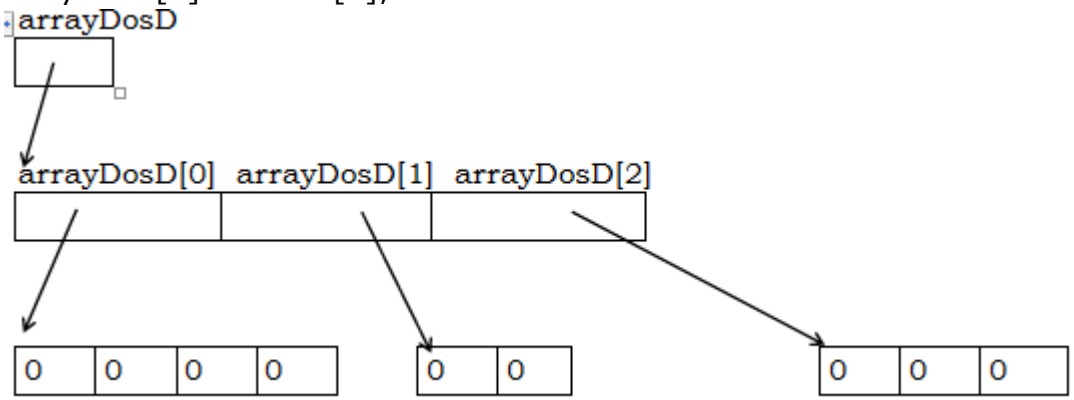
```
int[][] arrayDosD= new int[3][];
```

los corchetes vacíos hace que no se creen los arrays relativos a la segunda dimensión



a continuación podríamos crear individualmente cada array del segundo nivel con la posibilidad adicional de crear cada uno de un tamaño, con lo cual nos alejamos de la 'típica' visión de matriz matemática regular:

```
arrayDosD[0]=new int[4];
arrayDosD[1]=new int[2];
arrayDosD[2]=new int[3];
```



si visualizamos la estructura anterior de forma tabular entendemos ahora el concepto de array bidimensional irregular

0	0	0	0
0	0		
0	0	0	

Ejemplo: Ejecuta el siguiente código de forma que las matrices se carguen con valor *i*j +i +j* (no es importante el significado de la expresión anterior, es simplemente para dar un valor). Imprime el contenido. Como los arrays de la segunda dimensión son irregulares utilizamos bucles independientes para cada array de enteros. En el siguiente ejercicio, mejoraremos este código.

```
class Unidad4 {
  public static void main(String args[]){

      int[][] arrayDosD= new int[3][];
      arrayDosD[0]=new int[4];
      arrayDosD[1]=new int[2];
      arrayDosD[2]=new int[3];
      System.out.println("cargamos e imprimimos arrayDosD[0]. Observa que su  tamaño es 4");
      for(int j=0;j<4;j++){   //utilizamos la variable j pero podría ser i, z, ....
        arrayDosD[0][j]=0*j + 0 + j*2; //cargamos la matriz
```

```
      System.out.println("arrayDosD[0]["+ j +"]="+ arrayDosD[0][j]);
    }

    System.out.println("\ncargamos e imprimimos arrayDosD[1]. Observa que su  tamaño es 2");
    for(int j=0;j<2;j++){
      arrayDosD[1][j]=1*j + 1 + j*2; //cargamos la matriz
      System.out.println("arrayDosD[1]["+ j +"]="+ arrayDosD[1][j]);
    }

    System.out.println("\ncargamos e imprimimos arrayDosD[2]. Observa que su  tamaño es 3");
    //para ahondar en el concepto de referencia a array hago la ultima impresión con paso intermedio
    int[] x=arrayDosD[2];
    for(int j=0;j<3;j++){
      x[j]=2*j + 2 + j*2; //cargamos la matriz
      System.out.println("arrayDosD[2]["+ j +"]="+ x[j]);
    }
  }
}
```

El atributo length

Ya vimos en el boletín anterior que usar el atributo length es preferible a usar un literal entero para referirnos al tamaño de un array ya que hace el código más genérico y resistente a cambios. Para entender bien cómo aplicar length lo mejor es tener clara la forma como java almacena los arrays multidimensionales.

Ejercicio U4_B4A_E1: Volver a escribir el ejemplo anterior utilizando el atributo length, lo que permite utilizar un bucle anidado en otro y hacer un código más compacto.

Atributo length y matrices cuadradas.

Una conclusión evidente, es que si las matrices son cuadradas (mismo número de filas que de columnas), el tamaño de todos los *subarrays(filas)* es el mismo

```
class Unidad4 {
   public static void main(String args[]){

      int[][] arrayDosD= new int[2][2];
      arrayDosD[0][0]=1;
      arrayDosD[0][1]=2;
      arrayDosD[1][0]=3;
      arrayDosD[1][1]=4;
      System.out.println(arrayDosD.length);
      System.out.println(arrayDosD[0].length);
      System.out.println(arrayDosD[1].length);

   }
}
```

Y por tanto cualquiera de las expresiones anteriores es equivalente y se puede utilizar para controlar recorrido del array, por ejemplo

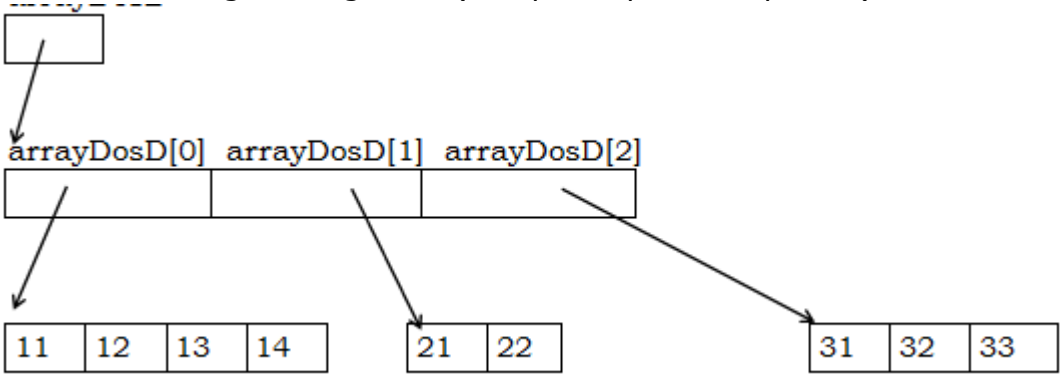
```
class Unidad4 {
   public static void main(String args[]){

      int[][] arrayDosD= new int[2][2];
      arrayDosD[0][0]=1;
      arrayDosD[0][1]=2;
      arrayDosD[1][0]=3;
      arrayDosD[1][1]=4;
      for(int i=0;i<arrayDosD.length;i++)
         for(int j=0;j<arrayDosD[i].length;j++)
            System.out.println(arrayDosD[i][j]);

      //como la matriz es cuadrada arrayDosD.length es equivalente a arrayDosD[i].length
```

```
        for(int i=0;i<arrayDosD.length;i++)
            for(int j=0;j<arrayDosD.length;j++)
                System.out.println(arrayDosD[i][j]);

    }
}
```

En general, es preferible el primer método, ya que por ejemplo la matriz podría dejar de ser cuadrada...

Inicializar un array de dos dimensiones al mismo tiempo que se crea
Similar a una dimensión, con los {}
por ejemplo, `int[][] matriz ={{0,2,4},{1,3,5}};`

Ejercicio U4_B4A_E2: Inicializa arrayDosD con la sintaxis anterior para que contenga el contenido del siguiente gráfico (e imprime para comprobar):



Ejercicio U4_B4A_E3: Si no lo hiciste ya, vuelve a escribir el ejercicio anterior de forma que utilice el "for mejorado". Puedes simplificar la impresión a algo del estilo de

```
11 12 13 14
21 22
31 32 33
```

Ejemplo de for+arrays dos dimensiones +break +etiquetas
Ahora que conocemos los arrays de dos dimensiones podemos ver este ejemplo del propio oracle en el que se aprecia el funcionamiento del break con etiquetas. En su momento como nos faltaban recursos el ejemplo de break con etiquetas era un poco "forzado" e "inútil".

Se trata de saber si un valor entero existe en una matriz de valores enteros y si existe una vez encontrado salir inmediatamente de la búsqueda (imagina que el conjunto de valores es muy grande y es importante la eficiencia)

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html

```
class BreakWithLabelDemo {
    public static void main(String[] args) {

        int[][] arrayOfInts = {
            { 32, 87, 3, 589 },
            { 12, 1076, 2000, 8 },
            { 622, 127, 77, 955 }
        };
```

```
        int searchfor = 12;

        int i;
        int j = 0;
        boolean foundIt = false;

    search:
        for (i = 0; i < arrayOfInts.length; i++) {
            for (j = 0; j < arrayOfInts[i].length;
                 j++) {
                if (arrayOfInts[i][j] == searchfor) {
                    foundIt = true;
                    break search;
                }
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor + " at " + i + ", " + j);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

Como siempre, sabiendo que el ejemplo anterior está escrito por un programador pata negra, aprovechamos para fijarnos en detallitos:

- Las impresiones sencillas, con println, sin rollos de printf o similares
- La inicialización del array se leería peor si fuera todo en una línea
- En este caso va bien definir i y j fuera de los bucles para poder consultar su valor al finalizar los bucles
- Y lo más importante para este boletín, fíjate cómo decide manejar el atributo length, escribe:
    - *arrayOfInts.length* no usa el literal entero *3*. Observa que 3 es menos claro, ya que tengo que pararme aunque sólo sea un segundo a comprobar que se corresponde con el tamaño de la primera dimensión. Si veo 3 no puedo deducir automáticamente que se refiere al tamaño ya que podría ser una matriz de 6 filas y por la razón que fuera querer recorrer sólo 3.
    - arrayOfInts[i].length no 4

Por tanto, Usar length aumenta la claridad y calidad del código

Ejercicio U4_B4A_E4:
Con una matriz cuadrada, introduciendo el tamaño por argumento, inicializar aleatoriamente la matriz con números de 0 a 99 y luego modificarla intercambiando la diagonal principal con la secundaria. Ejemplo:

```
L:\Programacion>java Unidad4 4
63      41      80      81
48      74      6       84
54      2       42      7
42      69      80      49


81      41      80      63
48      6       74      84
54      42      2       7
49      69      80      42
```

Es un problema similar a los que hicimos de imprimir asteriscos con bucles en el sentido que se trata de buscar la relación entre índices. Observa que al trabajar sólo con diagonales nos llega pensar sólo en función de la primera dimensión
La relación de índices para conseguir la nueva matriz es la siguiente
Para cada fila i
Matriz[i][i] intercambiar con matriz[i][tam-1-i];

Ejercicio U4_B4A_E5:
Con una matriz cuadrada, introduciendo el tamaño por argumento, inicializar aleatoriamente la matriz con enteros de 0 a 99 y luego modificarla invirtiendo los valores de la diagonal principal entre ellos, y los de la diagonal secundaria entre ellos como ilustra el ejemplo

```
L:\Programacion>java Unidad4 4
38      98      88      20
81      43      46      79
46      87      83      87
2       81      92      11


11      98      88      2
81      83      87      79
46      46      43      87
20      81      92      38
```

Relación indices
Para cada fila i hasta que i<tam/2:
En diagonal principal matriz[i][i]<=>matriz[tam-1-i][tam-1-i];
En diagonal secundaria matriz[i][tam-1-i]<=>matriz[tam-1-i][i];

Arrays de tres o más dimensiones
Tres o más dimensiones son más infrecuentes que una o dos dimensiones pero también se usan.

```
class Unidad4{
```

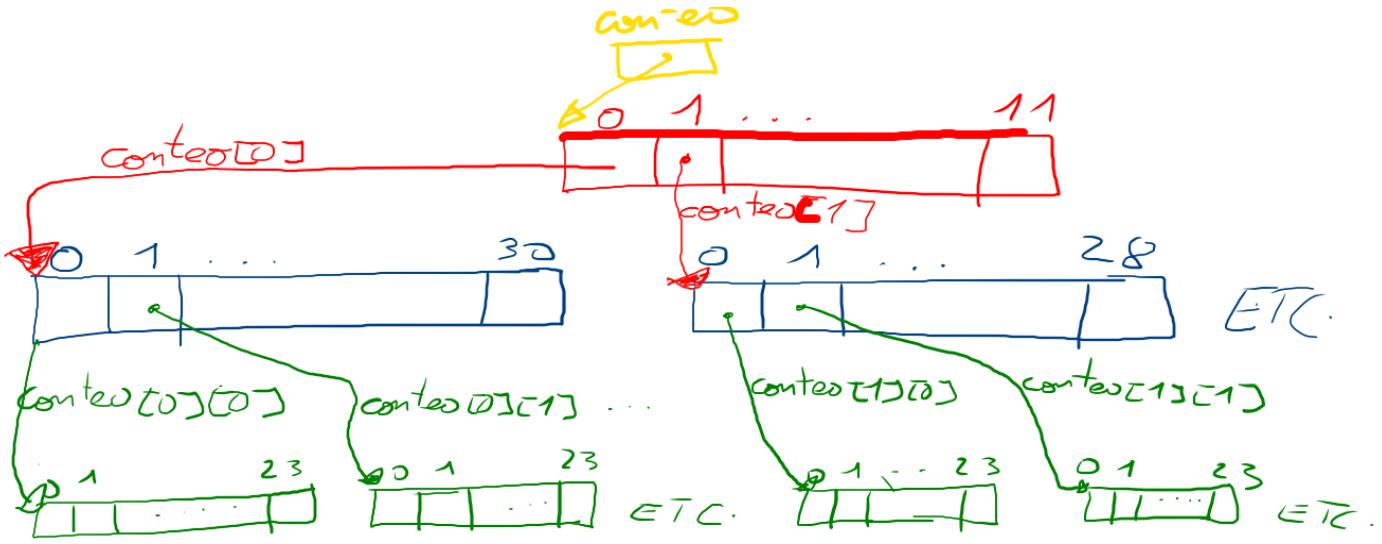
```
   public static void main(String args[]){
      int[][][] arrayTresD=  new  int[4][3][2];

      for(int i=0;i<4;i++)
         for(int j=0;j<3;j++)
            for(int k=0;k<2;k++){
               arrayTresD[i][j][k]=i+j+k;
               System.out.print(arrayTresD[i][j][k] + "  ");
            }
   }
}
```

Ejemplo:

Supongamos que estamos realizando un "conteo de coches", es decir, que estamos contando los coches que pasan por un determinado lugar en un periodo de tiempo que puede ser un día, varios días, varios meses, etc. La forma de declarar esos arrays podría ser la siguiente:

Duración del conteo	**Tipo de array**	**Declaración con Java (nc es Número de coches)**
Un día	Array de una dimensión (hora)	int[] nc = new int[24];
Varios días(1 mes por ejemplo. Suponemos mes=31 días)	Array de dos dimensiones (hora y día)	int[][] nc = new int[31][24];
Varios meses(1 año por ejemplo.)	Array de tres dimensiones (hora, día y mes)	int[][][] nc = new int[12][31][24];
Varios años(1 siglo por ejemplo.)	Array de cuatro dimensiones (hora, día, mes y año)	Int[][][][] nc = new int[100][12][31][24];
Varios siglos	Array de cinco dimensiones (hora, día, mes, año y siglo)	Int[][][][][] nc = new int[21][100][12][31][24];

Ejercicio U4_B4A_E6: Como caso concreto del cuadro anterior, se pide que simules un conteo con 3 dimensiones: hora, día y mes. Cada mes tendrá los días que naturalmente le corresponda (31, 30 o 28), para simplificar, despreciamos años bisiestos de forma que febrero siempre tiene 28 días(en el dibujo de abajo debería poner 27 en lugar de 28 para febrero). Llenamos aleatoriamente el conteo con números de coches entre 0 y 999. Podemos hacer esto por ejemplo con el siguiente código:




```
import java.util.Random;

public class Unidad4 {

   static int calcularDiasMes(int month) {
        int numDays = switch (month) {
           case 1,3,5,7,8,10,12->31;
           case 4,6,9,11->30;
           case 2->28;
           default->99;//mes erróneo
        };

        return numDays;

   }


    public static void main(String[] args) {
        //creamos array [mes][dia][hora]
        int[][][] conteo = new int[12][][];
        for (int mes = 0; mes < conteo.length; mes++) {
            //cada mes tiene un número de dias diferentes
            //se van creando las dimensiones dia y hora mes a mes
            conteo[mes] = new int[calcularDiasMes(mes + 1)][24];
        }

        //inicializamos el array. suponemos conteo entre 0 y 999
        Random r = new Random();
        for (int mes = 0; mes < conteo.length; mes++) {
            for (int dia = 0; dia < conteo[mes].length; dia++) {
                for (int hora = 0; hora < conteo[mes][dia].length; hora++) {
                    conteo[mes][dia][hora] = r.nextInt(1000);
                }
            }
        }
        imprimirConteos(conteo);


    }

}
```

Se pide completar el código escribiendo el método imprimirConteos()

un char puede indexar un array.

El siguiente ejercicio(*Ejercicio U4_B4A_E7: BANNER*) se puede resolver, entre otras formas, usando como índice de un array un char. Recuerda que un char es en muchos casos tratado como un caso especial de int sin signo y de 16 bits cuyo valor viene determinado por el código de caracteres.
Recuerda también que podemos asignar directamente a una variable entera un literal char ya que se hace un cast automático. En el siguiente ejemplo, x1 y x2 tienen ambas el valor 97

```
int x1='a';
int x2=(int)'a';
```

Los índices de los arrays son de tipo int. Si para el valor del índice en lugar de un int uso un char ocurre lo mismo que en las sentencias de arriba, un cast.

Ejemplo:

```
class Persona{
   String nombre;
   int edad;

   public Persona(String nombre, int edad) {
      this.nombre = nombre;
      this.edad = edad;
   }
 @Override
   public String toString() {
      return nombre+","+edad;
   }


}
class Unidad4{

   public static void main(String[] args) {
      //tamaños 256 si espero trabajar con caracteres con códigos entre 0 y 255
      //cada carácter tendrá asociada una posición en el array
      int[] a= new int[256];
      boolean[] b= new boolean[256];
      Persona[] c= new Persona[256];

      //el caracter b tiene como código entero 98
      a['b']=99999;//equivalente a  a[98]=99999;
      System.out.println(a[98]+ "   "+ a['b']+  " "+ a[(int)'b']);

      //la A tiene como código entero 65
      b['A']=false;
      System.out.println(b[65]+ "   "+ b['A']+  " "+ b[(int)'A']);
      //la a tiene como código entero 97
      c['a']=new Persona("yo",50);
      System.out.println(c[97]+ "   "+ c['a']+  " "+ c[(int)'a']);

   }
}
```

Observa que indicar entre las llaves de array un carácter es lo mismo que indicar el entero asociado a dicho carácter.

Y por tanto tenemos tres formas de indexar por ejemplo la posición 98 de un array

```
A[98]    a['b']     a[(int)'b']
```

Ejemplo: dado un string comprobamos la frecuencia de aparición de caracteres de dicho string. Suponemos que el string sólo tiene caracteres con códigos entre 0 y 255

```
class Unidad4{

   public static void main(String[] args) {
      //suponemos caracteres con códigos entre  0 y  255
      String s= "El perro de san roque no tiene rabo";
      int[] frecuencias= new int[256];

      for(int i=0;i<s.length();i++){
         frecuencias[s.charAt(i)]++;
      }
      //imprimo  frecuencias de letras minúsculas
      for(int i='a';i<='z';i++){
         System.out.println((char)i+ " apareció "+ frecuencias[i]+ " veces");
      }


   }
}
```

Ejercicio U4_B4A_E7: BANNER

Se indica como argumento por línea de comandos un String. La salida imprime un banner para dicho string con el caracter # (u otro que te guste)
Hay una variedad de soluciones. Siempre que tenemos que manejar char[] podemos usar Strings pero procura solucionar el problema usando un char[][][] para practicar la multidimensión. Una ventaja de usar sólo char (sin Strings) es que las soluciones suelen ser más eficientes ya que son de más bajo nivel. Es una discusión similar a los problemas de bucles e impresión de figuras con asteriscos, es un reto hacerlos a la antigua, cuando no existía el recurso de String.

```
L:\Programacion>java Unidad4 "TELMA TELMAAAAAAAA"
 ##### ####  #     #   # ####        ##### ####  #     #   # ####  ####  ####  ####  ####  ####  ####  ####
   #   #     #     ## ## #  #          #   #     #     ## ## #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
   #   ###   #     # # # ####          #   ###   #     # # # ####  ####  ####  ####  ####  ####  ####  ####
   #   #     #     #   # #  #          #   #     #     #   # #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
   #   ####  ####  #   # #  #          #   ####  ####  #   # #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #

L:\Programacion>
```

Para almacenar el dibujo de un carácter puedes usar una matriz de char[][] por ejemplo, almacenamos el dibujo de la letra C como un array de arrays de char

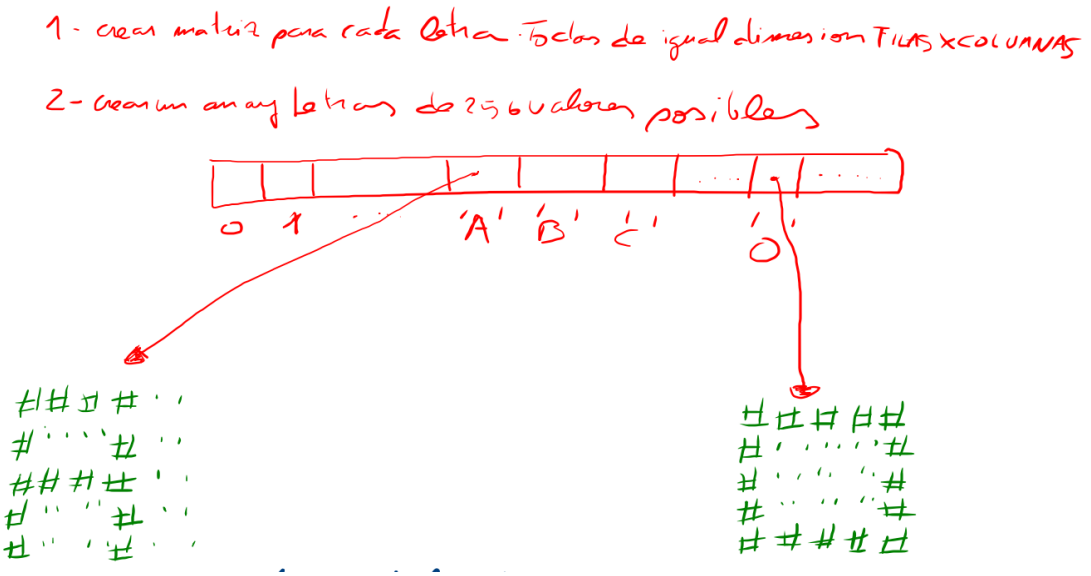
```
char[][] C = {{'#', '#', '#', '#',' '},
              {'#', ' ', ' ', ' ',' '},
              {'#', ' ', ' ', ' ',' '},
              {'#', ' ', ' ', ' ',' '},
              {'#', '#', '#', '#',' '}
```

No es necesario que manejes muchos caracteres, con unos cuantos para hacer la pruebas que quieras es suficiente.
Observa que el banner hay que imprimirlo linea a linea y tenemos que ir imprimiendo secuencialmente:

- La primera de línea de todos los caracteres
- La segunda línea de todos n caracteres
- …
- La quinta línea de todos los caracteres (suponiendo que todos los caracteres van a estar formados por 5 líneas)

Un mecanismo de evitar este condicionante de la consola que "no nos dejar regresar linea arriba" es crear un String que contenga el banner final y finalmente imprimir este String, pero es un reto mayor no usar String en ningún caso. En todo caso, si finalmente no te sale con char[][][] puedes usar Strings como mejor te convenga.

Un posible enfoque para resolver el problema de Banner con char[][][]



Hay muchas formas de razonar y resolver este problema, esta es una. Cada carácter está almacenado en una matriz. Todos los caracteres están almacenados en matrices con las mismas dimensiones. Si los caracteres por ejemplo tiene 5 filas lo que hago es primero imprimir la 1º fila de TODOS los caracteres de la frase, luego la 2º etc.

```
Para cada fila
        Para cada caracter c de la frase
                Para cada columna de la fila
                        Imprimir letras[c][fila][columna]
```

Un par de retos muy asequibles

Los retos matriz identidad y matrices triangulares son muy asequibles y vienen en la categoría de arrays multidimensionales

Categoría Arrays multidimensionales

Las soluciones de estos problemas necesitan manejar arrays de dos o más dimensiones.

Problemas

Id	Título
101	Cuadrados diabólicos y esotéricos
129	Marcadores de 7 segmentos
137	Hundir la flota
151	¿Es matriz identidad?
160	Matrices triangulares

Ejercicio U4_B4A_E8: Matriz identidad acepta el reto
https://www.aceptaelreto.com/problem/statement.php?id=151&cat=14

Ejercicio U4_B4A_E9: Acepta el reto.Matrices Triangulares id 160

https://www.aceptaelreto.com/problem/statement.php?id=160

Problema de matriz que no se resuelve con matriz

Observa el siguiente problema

https://www.aceptaelreto.com/problem/statement.php?id=244

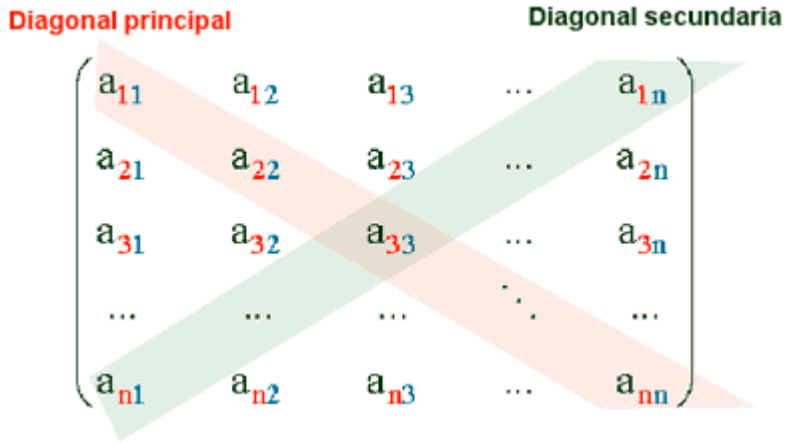
Un tablero de ajedrez es evidente que se puede representar muy puramente con un array de dos dimensiones. Por tanto, el primer impulso sería crear una matriz bidimensional y luego hacer las comprobaciones correspondientes de filas, columnas y diagonales. Pero esto es laborioso y poco eficiente. Este es un problema bien conocido y muy documentado en internet y aunque resulte chocante en un primer momento ino se usa una matriz (array de dos dimensiones) para solucionar el problema!

El algoritmo.

Hay muchas variantes. Algunos con relaciones matemáticas intrincadas. Nosotros usaremos uno sencillo, quizá no muy eficiente, con el que basta almacenar las coordenadas de las reinas en dos arrays paralelos. Una reina ataca a otra si está en la misma fila, columna o diagonal. Para decidir si dos reinas están en la misma diagonal basta con fijarse que:

- en las diagonales principales, siempre ocurre que si dos reinas i y j están en la misma diagonal principal entonces la resta de las coordenadas coincide
- En las diagonales secundarias lo que coincide es la suma de las coordenadas.

Llamamos diagonales principales a la diagonal principal y todas sus paralelas. Similar para diagonales secundarias. Puedes comprobar en el siguiente gráfico las relaciones aritméticas entre índices en las diagonales.



Un pseudocódigo

1. leer las coordenadas de las reinas y almacenarlas secuencialmente en dos arrays paralelos cx y cy. cx almacena coordenadas columna y cy filas..
2. Se examinan las coordenadas comparando todas con todas en bucle anidado. La primera se compara con todas las demás, y así sucesivamente con la segunda, tercera … hasta encontrar un ataque en cuyo caso salimos de todo bucle. Esta búsqueda de ataque se puede mejorar fácilmente ya que por ejemplo la tercera si llegamos a ella sabemos que la primera y segunda no detectaron ataque con ella.

```
hayAtaque=false
bucleExterior:
para i =0; i<r-1;i++
        //comparamos reina i con las que vienen despues
          para j=i+1;j<r;j++
                si cx[i]==cx[j] hayAtaque en columna
                si cy[i]==cy[j] hayAtaque en fila
                si cx[i]-cy[i] ==   cx[j]-cy[j] hayAtaque en diagonal principal
                si cx[i]+cy[i] ==   cx[j]+cy[j] hayAtaque en diagonal secundaria
                si hay ataque break bucleExterior

finpara
si hayAtaque imprimir SI sino NO
```

Por claridad en el pseudocódigo anterior se utilizaron IFs secuenciales, pero sería más eficiente hacerlo todo con un único if y encadenando las condiciones con el or perezoso ||

Ejercicio U4_B4A_E10: en coderunner resolver reinas atacadas