TEMA 3. ALMACENAMIENTO DE LA INFORMACIÓN EN ESTRUCTURAS DE DATOS

INDICE

ARRAYS. DEFINICIÓN

Un array o vector (o arreglo) es una colección de variables del mismo tipo que ocupan posiciones contiguas de memoria y que se referencian por un nombre en común que es a su vez una variable de tipo puntero que contiene la dirección donde está situado el primer elemento del array en memoria.

A un elemento específico de un array se accede mediante un **índice** que va de **0 a N-1** siendo N el número de elementos de que consta el array. La dirección más baja de memoria corresponde al primer elemento. Al acceder a los elementos de un array hay que tener mucho cuidado de no rebasar el índice del último elemento ya que estaríamos provocando un acceso a un área de memoria a la que no tenemos permiso para acceder lo que podría provocar un error de protección de memoria.

La cantidad de memoria requerida para almacenar un array está directamente relacionada con su tipo y su tamaño.

Cuando la posición de los elementos de un array tiene relación con el valor que se almacena en ella se dice que el array es **asociativo**.

Los arrays pueden tener una o varias dimensiones.

DECLARACION DE ARRAYS. PSEUDOCODIGO

Para declarar un array hay que indicar el tipo de dato del array, el nombre del array y, entre corchetes, el tamaño del array.

Por ejemplo, si queremos definir un array de 5 posiciones de tipo ENTERO y de nombre ARRAYENTEROS escribimos

```
ENTERO ARRAYENTEROS[5]
```

DECLARACION DE ARRAYS. JAVA

Para declarar un array en java hay que indicar el tipo de dato del array seguido de unos corchetes y el nombre del array. Una vez declarado el array tenemos que crearlo usando la función new, dentro de la cual indicamos el tamaño del array.

El proceso se puede realizar en uno o dos pasos.

Por ejemplo, si queremos definir un array de **5** posiciones de tipo **int** y de nombre **arrayenteros** escribimos

```
int [] arrayenteros;
arrayenteros = new int [5];
o
int [] arrayenteros = new int [5];
```

RECORRIDO DE ARRAYS

El recorrido de un array se usa para inicializar los valores de un array (por ejemplo para poner el valor 0 en todas las posiciones), para buscar un determinado valor,...

A la hora de recorrer un array tenemos varias posibilidades:

Recorrido completo. La operación de recorrido completo de un array consiste en ir desde la primera posición (de índice 0) hasta la última (de índice n-1). La estructura que se utiliza normalmente es una repetitiva de tipo N veces que va desde 0 hasta n-1.

Recorrido de los elementos útiles del array. Cuando trabajamos con arrays debemos distinguir entre el tamaño real del array y el número de posiciones que estamos realizando realmente. En este caso necesitamos dos variables, una para contener el tamaño del array (que puede cambiar) y otra para contener el número de elementos útiles que tiene el array. Por ejemplo, el siguiente array tiene un tamaño de 5 pero sólo están siendo utilizadas 2 posiciones

0	1	2	3	4
2	4			

AÑADIR UN ELEMENTO A UN ARRAY

Antes de añadir un elemento a un array debemos comprobar que el array no esté lleno. Si el array está lleno deberemos aumentarlo de tamaño.

A la hora de añadir un elemento a un array tenemos varias posibilidades:

Añadir al final. Para ello basta con colocar el elemento en la posición siguiente a la última utilizada. Este valor debe de estar almacenado en una variable. Por ejemplo, si en el array anterior queremos añadir el valor 3 al final deberemos tener una variable que contenga el valor del número de elementos útiles. El valor 3 se añadirá en la posición correspondiente al número de elementos. Al añadir el nuevo elemento el valor de esa variable se incrementará en 1 (en este caso pasará de valer 2 a valer 3)

0	1	2	3	4
2	4	3		

Añadir en el medio conociendo la posición. Para ello, antes de colocar el elemento en la posición deseada deberemos mover todos los valores desde esa posición hasta el final una posición a la derecha empezando por el último. Por ejemplo, añadimos al array anterior el valor 6 en la posición 0. Al añadir el nuevo elemento el número de elementos se incrementará en 1 (en este caso pasará de valer 3 a valer 4)

0	1	2	3	4
6	2	4	3	

Añadir de manera ordenada. Para añadir de manera ordenada vamos a usar el **método de inserción o de la baraja**. Para ello, antes de colocar el elemento en la posición que le corresponda deberemos buscar cual es esa posición. Una vez encontrada la posición, deberemos mover todos los valores desde esa posición hasta el final una posición a la derecha empezando por el último. Para finalizar, guardamos el nuevo elemento en la posición. Por ejemplo, vamos añadir al siguiente array el valor 3.

0	1	2	3	4
2	4	6	8	

En primer lugar realizamos un recorrido para buscar la posición en la que vamos a colocar el nuevo elemento. En este caso la posición 1.

Una vez encontrada la posición desplazamos los elementos desde esa posición hasta el final una posición a la derecha.

0	1	2	3	4
2	4	4	6	8

Colocamos el valor del elemento en esa posición.

0	1	2	3	4

2	3	4	6	8

Al añadir el nuevo elemento el número de elementos se incrementará en 1 (en este caso pasará de valer 4 a valer 5)

BORRAR UN ELEMENTO DE UN ARRAY

A la hora de añadir un elemento a un array tenemos varias posibilidades:

Borrar marcando el elemento. Para ello basta con colocar en la posición del elemento a borrar un valor que identifica de manera inequívoca a los elementos borrados (un valor que no vayan a tener los elementos útiles). Este método desperdicia espacio en memoria ya que no se modifica el número de posiciones ocupadas del array por lo que no se recomienda. Por ejemplo, para borrar el elemento de valor 4 en el siguiente array ponemos el valor -1 como marca de borrado

0	1	2	3	4
2	3	4	6	

Al borrar quedaría

0	1	2	3	4
2	3	**-1**	6	

Borrar compactando el array. Este método sirve tanto si la posición es conocida como si la tenemos que buscar. Una vez que conocemos la posición del elemento a borrar debemos mover todos los valores desde esa posición hasta el final una posición a la izquierda. Por ejemplo, vamos a borrar del siguiente array el valor 3.

0	1	2	3	4
2	3	4	6	

En primer lugar realizamos un recorrido para buscar la posición en la que se encuentra el elemento a borrar. Si no se encuentra el elemento no se puede borrar. En este caso lo hemos encontrado en la posición 1.

Una vez encontrada la posición desplazamos los elementos desde esa posición hasta el final una posición a la izquierda.

0	1	2	3	4
2	4	6	6	

Al desplazar los elementos una posición a la izquierda en el array sigue estando el último valor útil anterior. Aunque aparezca ya no es un elemento útil por lo que para finalizar el proceso de borrado debemos disminuir el número de elementos en 1 (en este caso pasará de valer 4 a valer 3).

EJERCICIOS DE ARRAYS NUMÉRICOS. PSEUDOCODIGO

1. Realiza el pseudocódigo para el programa NUMVECES que pide números de 0 a 9 por pantalla y calcula el número de veces que han sido introducidos utilizando un array. Finaliza la introducción de datos cuando se introduce un valor negativo. Al finalizar la introducción de datos muestra por pantalla el número de veces que se ha introducido cada número.

2. Realiza el pseudocódigo para el programa MEDIANUM que pide números por pantalla y los almacena en un array hasta que se introduzca un número negativo que no será tenido en cuenta o hasta que esté lleno el array. Después calcula la Media y la muestra.

3. Realiza el pseudocódigo para el programa MEDIATEM que pide temperaturas por pantalla y las almacena en un array hasta que se introduzca la temperatura -999 que no será tenida en cuenta o hasta que esté lleno el array. Después calcula la temperatura media y muestra el número de temperaturas que son mayores que la media, iguales que la media, e inferiores a la media.

4. Realiza el pseudocódigo para el programa ARRAYSUMAPRIMO que pide números enteros por pantalla y los almacena en un array hasta que se introduzca un número negativo que no será tenido en cuenta o

hasta que esté lleno el array. Después calcula la suma de todos ellos y comprueba si es un número primo o no.

EJERCICIOS DE ARRAYS NUMÉRICOS. JAVA

5. Realiza la clase Java Numveces que pide números de 0 a 9 por pantalla y calcula el número de veces que han sido introducidos utilizando un array. Finaliza la introducción de datos cuando se introduce un valor negativo. Al finalizar la introducción de datos muestra por pantalla el número de veces que se ha introducido cada número.

6. Realiza la clase Java Medianum que pide números por pantalla y los almacena en un array hasta que se introduzca un número negativo que no será tenido en cuenta. Después calcula la Media y la muestra.

7. Realiza la clase Java Mediatem que pide temperaturas por pantalla y las almacena en un array hasta que se introduzca la temperatura -999 que no será tenida en cuenta. Después calcula la temperatura media y muestra el número de temperaturas que son mayores que la media, iguales que la media, e inferiores a la media.

8. Realiza la clase Java Arraysumaprimo que pide números enteros por pantalla y los almacena en un array hasta que se introduzca un número negativo que no será tenido en cuenta. Después calcula la suma de todos ellos y comprueba si es un número primo o no.

ARRAYS DE CARACTERES Y STRINGS

En Java los arrays de caracteres y los Strings son dos tipos distintos de datos. Los arrays de caracteres se crean igual que los arrays de datos de tipo entero.

Para crear un String **NO** basta con definir la variable de tipo String

```
String cadena;
```

También debemos indicar que lo cree. Para ello dedemos escribir

```
String cadena = ""; o
String cadena = new String();
```

De esta manera lo que creamos es un String nulo. Podemos crear un String asignándole un valor inicial. Para ello debemos escribir

```
String cadena = "Hola Mundo."; o
String cadena = new String("Hola Mundo.");
```

OPERACIONES CON STRINGS

En Java las operaciones que se pueden realizar con Strings y las operaciones que se pueden realizar con arrays de caracteres no son las mismas. Las operaciones que se pueden realizar sobre los arrays de caracteres son las mismas que se pueden realizar sobre los arrays de datos de tipo entero.

Si queremos que un String se comporte como un array de caracteres debemos de crear un array de caracteres con el contenido del String y usarlo para realizar las operaciones.

Obtener el valor de una posición del String. Para obtener el valor que hay en una posición de un String debemos usar el método **charAt(posición)**. Por ejemplo, si queremos obtener el valor de la posición 0 de un String de nombre cadena escribimos

```
caracter = cadena.charAt(0);
```

Dar un valor a una posición del String. Para dar un valor a una posición de un String debemos convertir primero el String a un **array de caracteres** (**char[]**), después cambiar el valor del carácter de esa posición por el deseado y, finalmente convertir el array de caracteres a String y asignarselo de nuevo al String original. Por ejemplo, si queremos poner el valor 'A' en la posición 0 de un String de nombre cadena escribimos

```
String cadena = "Hola Mundo";
char[] cadenatemp = cadena.toCharArray();
cadenatemp[0] = 'A';
cadena = new String(cadenatemp);
```

Poner en mayúsculas todos los caracteres de un String. Para poner en mayúsculas todos los caracteres del String se utiliza la función **toUpperCase**. Por ejemplo, si queremos poner en mayúsculas todos los caracteres del String de nombre cadena escribimos

```
cadena = cadena.toUpperCase ();
```

Poner en minúsculas todos los caracteres de un String. Para poner en minúsculas todos los caracteres del String se utiliza la función **toLowerCase**. Por ejemplo, si queremos poner en minúsculas todos los caracteres del String de nombre cadena escribimos

```
cadena = cadena.toLowerCase ();
```

Obtener el número de caracteres de un String (longitud). Para obtener el número de caracteres de un String se utiliza la función **length**. Por ejemplo, si queremos obtener el número de caracteres del String de nombre cadena escribimos

```
int longitud = cadena.length();
```

COMPARACION IGUALDAD DE DATOS DE TIPO STRING

Uno de los errores más comunes a la hora de comparar datos de tipo String es el de utilizar el operador de igualdad (**==**) cuando se tiene que usar la función **equals** y viceversa.

Comparador de igualdad (==). Se usa cuando se quiere comparar si dos variables de tipo String hacen referencia al mismo objeto. En el siguiente caso las dos variables de tipo String s1 y s2 hacen referencia al mismo objeto

```
String s1 = "Hola Mundo.";
String s2 = s1;
if (s1 == s2){
 // sería true en este caso
}
```

```
String s1= "Hola Mundo.";
```

```
String s2 = "Hola Mundo.";
if (s1 == s2){
 // sería false en este caso
}
```

Función equals. Se usa cuando se quiere comparar si dos variables de tipo String tienen el mismo contenido. En el siguiente caso las dos variables de tipo String s1 y s2 son distintas pero tienen el mismo contenido

```
String s1= "Hola Mundo.";
String s2 = new String("Hola Mundo.");
if(s1.equals(s2)){
 // sería true en este caso
}
if(s1.equals("")){
 // sería false en este caso
}
if(s1.equals("Hola Mundo.")){
 // sería true en este caso
}
```

Cuando las variables de tipo String s1 y s2 hacen referencia al mismo objeto el contenido de ambas es el mismo (*"Hola Mundo."* en nuestro caso). El matiz es que el hecho de que dos variables de tipo String tengan el mismo contenido no implica que ambas variables hagan referencia al mismo objeto de tipo String. Este matiz ocasiona muchos problemas a la hora de comparar datos de tipo String.

COMPARACIONES DE DATOS DE TIPO STRING

Para la comparación alfabética del contenido de los datos de tipo String se usa la función **compareTo(String)**. La función compareTo devuelve un valor entero menor que cero si el String que realiza la comparación es menor que el String que se pasa como parámetro, un valor entero mayor que cero si el String que realiza la comparación es mayor que el String que se pasa como parámetro, y devuelve cero si el contenido de ambos Strings es el mismo. Por ejemplo si

```
String s1= "Hola Mundo"; String s2 = "Adios"; s1.compareTo(s2)      Devuelve un entero mayor que cero
String s1= "Adios"; String s2 = "Hola Mundo"; s1.compareTo(s2)      Devuelve un entero menor que cero
String s1= "Hola Mundo"; String s2 = "Hola Mundo"; s1.compareTo(s2) Devuelve cero
```

CONVERTIR UN VALOR NUMERICO A UN VALOR DE TIPO STRING

En algunas ocasiones es necesario convertir un valor numérico a un valor de tipo String. Para ello se usa la función **String.valueOf(valor)**. Esta función recibe como valor un dato de tipo entero (int, long, ...) o de tipo real (float, doublé,...) y devuelve un dato de tipo String. Por ejemplo

```
String s = String.valueOf(5);      Asigna a la variable de tipo String s el valor "5"
String s = String.valueOf(3.14);   Asigna a la variable de tipo String s el valor "3.14"
```

CONVERTIR UN VALOR DE TIPO STRING A UN VALOR NUMERICO

En algunas ocasiones es necesario convertir un valor de tipo String a un valor numérico. Antes de realizar la conversión es aconsejable borrar los espacios en blanco que pueda tener el String al principio y al final usando la función **trim()**.

Para convertir un dato de tipo String a otro de tipo int se usa la función **Integer.parseInt(String)**. Por ejemplo

*String s = " 5 "; **int n= Integer.parseInt(s.trim());** Asigna a la variable de tipo int n el valor entero 5*

Para convertir un dato de tipo String a otro de tipo double debemos realizar dos operaciones. Primero convertimos el dato de tipo String a un objeto de tipo **Double** usando la función **Double.valueOf(String)**. Después convertimos el objeto de tipo Double a un dato de tipo double usando la función **doubleValue()**. Es muy importante saber diferenciar entre un objeto de tipo Double y una variable de tipo doucble. Por ejemplo

String s = " 3.14 "; ***Double D = Double.valueOf(s.trim());*** *Asigna al objeto de tipo Double de nombre D el valor 3.14* ***double d = D.doubleValue();*** *Asigna a la variable de tipo double el valor 3.14*

Importante. Para que la conversión se realice correctamente el valor contenido en el String debe de ser un valor numérico válido. Si se intenta convertir a un valor numérico un String que no contiene un valor numérico válido se produce una excepción del tipo **java.lang.NumberFormatException**. Las excepciones de este tipo las controlaremos en el tema relativo a las excepciones.

EJERCICIOS DE ARRAYS DE CARACTERES Y STRINGS

9. Realiza la clase Java Contrase que pide un nombre por pantalla y una contraseña. Compara los valores con unos valores predefinidos y si coinciden muestra un mensaje de bienvenida. Si no coinciden muestra un mensaje de error por pantalla. El proceso se repite hasta que los datos son correctos o hasta que se produzcan más de 3 intentos.

10. Realiza la clase Java Caracmay pide un String por pantalla y calcula cual es el carácter o caracteres Mayores y devuelve el carácter y la posición que ocupa.

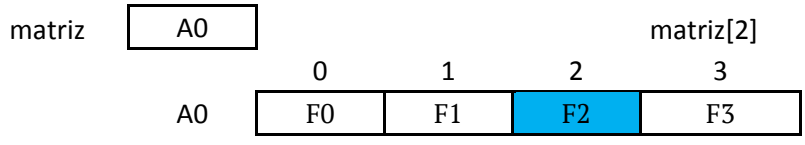
11. Realiza la clase Java Contavoc que solicita la introducción de una cadena por teclado. Cuenta el número de veces que se ha introducido cada una de las vocales sin diferenciar entre mayúsculas y minúsculas utilizando un array, y muestra el número de veces que se ha introducido cada una de las vocales.

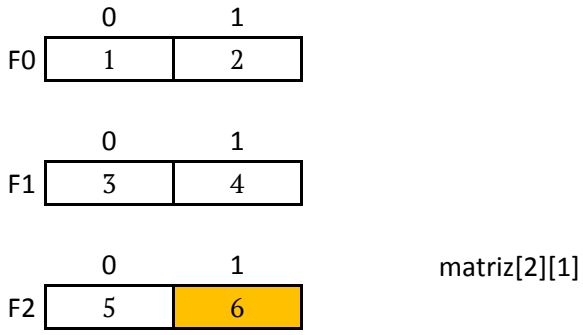
MATRICES NUMÉRICAS. DEFINICIÓN

Una matriz es un array bidimensional. Las matrices se componen de varias filas dentro de cada una de las cuales hay varias columnas. Internamente una matriz es un array unidimensional que tiene tantas columnas como filas tenga la matriz. Por cada fila de la matriz se crea un array y la dirección de memoria donde se encuentra ese array correspondiente a la fila se guarda en la columna correspondiente. Por ejemplo, si tenemos una matriz de 3 filas y 2 columnas y de nombre matriz nosotros la representamos

matriz	Columna 0	Columna 1
Fila 0	1	2
Fila 1	3	4
Fila 2	5	6

Pero internamente el sistema la representa





Es muy importante no olvidar esta estructura interna ya que la mayoría de los problemas que surgen al trabajar con matrices vienen derivados de no comprender completamente como se almacenan internamente en memoria.

Para definir una matriz en Java debemos especificar en primer lugar su tipo, después sus dimensiones mediante corchetes (**[]**) y para finalizar su nombre. Por ejemplo, para definir una matriz de datos de tipo int, con 3 filas y 2 columnas y de nombre matriz escribimos

```
int[][] matriz = new int[3][2];
```

En Java se pueden crear matrices con un número de columnas distinto para cada fila. En los casos en los que las filas no tienen el mismo número de columnas el proceso de creación de la matriz cambia. Por ejemplo, para definir una matriz de datos de tipo int y de nombre matriz con 3 filas en la que la fila 0 tiene 2 columnas, la fila 1 tiene 3 columnas y la fila 2 tiene 4 columnas escribimos

```
int[][] matriz = new int[3][];
matriz[0] = new int[2];    // la fila 0 tiene 2 columnas
matriz[1] = new int[3];    // la fila 1 tiene 3 columnas
matriz[2] = new int[4];    // la fila 2 tiene 4 columnas
```

También se pueden crear matrices usando la operación de inicialización.

INICIALIZACION DE MATRICES

Al crear una matriz el compilador le asigna a cada una de sus posiciones un valor por defecto que depende del tipo de dato de la matriz. Por ejemplo si el dato es un

Array numérico. El valor inicial de cada una de las posiciones es **0**.
Array de caracteres. El valor inicial de cada una de las posiciones es el carácter **'\u0000' (carácter nulo)**.
Array booleano. El valor inicial de cada una de las posiciones es **false**.
Array de Strings. El valor inicial de cada una de las posiciones es **null**.
Array de cualquier otro tipo de Objetos. El valor inicial de cada una de las posiciones es **null**.

Para inicializar una matriz debemos de conocer su número de filas y de columnas. Si todas las filas tienen el mismo número de columnas la inicialización se puede hacer usando un bucle anidado. Por ejemplo, para inicializar una matriz de datos de tipo int, con 3 filas y 2 columnas y de nombre matriz con el valor -1 escribimos

```
for(int fila = 0;fila < 3;fila++){
  for(int columna = 0; columna < 2; columna ++){
    matriz[fila][columna] = -1;       // pongo el valor -1 a la posición de la matriz
```

```
  }
}
```

También podemos asignar valores manualmente a cada una de las posiciones en el **momento de la creación de la matriz**. Esto suele ser útil cuando las posiciones tienen valores distintos o cuando las filas tienen distinto número de columnas. Por ejemplo, para inicializar una matriz de datos de tipo int con 3 filas y 2 columnas y de nombre matriz en el momento de su creación con valores distintos en cada una de sus posiciones escribimos

```
int [][] matriz = {{1,2}, {3, 4}, {5,6}};
```

Por ejemplo, para inicializar una matriz de datos de tipo int y de nombre matriz con 3 filas en la que la fila 0 tiene 2 columnas, la fila 1 tiene 3 columnas y la fila 2 tiene 4 columnas en el momento de su creación escribimos

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

RECORRIDO DE MATRICES

Para recorrer una matriz debemos de conocer su número de filas y de columnas. Si todas las filas tienen el mismo número de columnas el recorrido (aligual que la inicialización) se puede hacer usando un bucle anidado. Por ejemplo, para recorrer una matriz de datos de tipo int, con 3 filas y 2 columnas y de nombre matriz mostrando el contenido de cada fila en una línea escribimos

```
for(int fila = 0;fila < 3;fila++){
  for(int columna = 0; columna < 2; columna ++){
    System.out.print(matriz[fila][columna]);          // muestro por pantalla el valor de la posición de la matriz
  }
  System.out.println();     // salto a la siguiente línea
}
```

Si la matriz no tiene el mismo número de columnas en cada fila debemos usar el atributo **length** a la hora de realizar el recorrido. Por ejemplo, para recorrer una matriz de datos de tipo int y de nombre matriz con 3 filas en la que la fila 0 tiene 2 columnas, la fila 1 tiene 3 columnas y la fila 2 tiene 4 columnas mostrando el contenido de cada fila en una línea escribimos

```
for(int fila = 0;fila < 3;fila++){
  for(int columna = 0; columna < matriz[fila].length; columna ++){
    System.out.print(matriz[fila][columna]);          // muestro por pantalla el valor de la posición de la matriz
  }
  System.out.println();     // salto a la siguiente línea
}
```

Para evitar problemas podemos usar el atributo **length** tanto a la hora de recorrer las filas como a la hora de recorrer las columnas. Por ejemplo, para realizar el ejemplo anterior usando el atributo length también en las filas escribimos

```
for(int fila = 0;fila < matriz.length;fila++){
  for(int columna = 0; columna < matriz[fila].length; columna ++){
    System.out.print(matriz[fila][columna]);          // muestro por pantalla el valor de la posición de la matriz
  }
```

```
  System.out.println();     // salto a la siguiente línea
}
```

EJERCICIOS DE MATRICES NUMÉRICAS

12. Realiza la clase Java Arraysum que solicita la introducción de una matriz de números enteros por teclado, calcula la suma de los elementos de cada una de las filas y mete el resultado de ese cálculo en un array. Después visualiza el contenido del array por pantalla.

13. Realiza la clase Java Inicia2d que inicializa una matriz de caracteres de 2 dimensiones con el carácter # y después la muestra por pantalla.

14. Realiza la clase Java Traspues que pide la introducción de los datos de una matriz por pantalla, traspone la matriz, y muestra la matriz traspuesta por pantalla.

15. Realiza la clase Java Sumamat que solicita la introducción de dos matrices de números enteros por teclado, calcula la suma de las dos matrices en otra matriz, y muestra la matriz resultado por pantalla.

16. Realiza la clase Java Restamat que solicita la introducción de dos matrices de números enteros por teclado, calcula la resta de las dos matrices en otra matriz, y muestra la matriz resultado por pantalla.

17. Realiza la clase Java Arrayop que muestra el siguiente menú por pantalla para permitir realizar operaciones básicas con arrays.

```
MENU DE MANEJO DE ARRAYS
1. Inicializar el array
2. Ver el array
3. Insertar un elemento (en la posición que le corresponda)
4. Borrar un elemento (de su posición compactando el array)
5. Salir
```

18. Realiza la clase Java Multimat calcula la multiplicación de dos matrices bidimensionales A y B que tienen como característica principal que el número de columnas de A es igual que el número de filas de B y que la matriz resultado tiene tantas filas como A y tantas columnas como B. Cada uno de los términos de la matriz resultado se obtiene sumando la multiplicación de los elementos de una fila de A por los de una columna de B.

ARRAYS DE STRINGS

Los arrays de Strings se comportan igual que los arrays que contienen tipos de datos numéricos.

Por ejemplo, si queremos definir un array que contenga 5 datos de tipo String lo debemos definir de la siguiente manera

```
String[] arraycadenas = new String [5];
```

EJERCICIOS DE ORDENACION DE ARRAYS

19. Realiza la clase Java ArrayOrdenado que recibe números enteros positivos por teclado y los almacena en un array en orden ascendente hasta que llega un número negativo que no será tenido en cuenta.

20. Realiza la clase Java Diccionario que recibe palabras por teclado y las almacena en un array de Strings en orden alfabético ascendente hasta que llega una cadena vacía que no será tenida en cuenta (**isEmpty()**).

21. Realiza la clase Java DiccionarioInverso que recibe palabras por teclado y las almacena en un array de Strings en orden Descendente hasta que llega una cadena vacía que no será tenida en cuenta (**isEmpty()**).

22. Realiza la clase Java Fusionar que recibe dos arrays de cadenas de caracteres ordenados en ascendente y los fusiona en un tercero que también estará también ordenado en ascendente.