TEMA 4 : DATOS ESTRUCTURADOS

4.1 Arrays

4.2 Matrices

4.3 Operaciones con arrays

4.1.- Arrays.

Una estructura de datos es un conjunto finito y ordenado de elementos homogéneos que se caracteriza por su organización y por las operaciones que se definen en ella.

Los datos de tipo estándar (entero, real, cadena, lógico, carácter) se pueden organizar en diferentes estructuras de datos estáticas y dinámicas.

Las estructuras de datos estáticas son aquellas en las que el espacio ocupado en memoria se define en tiempo de compilación y no puede ser modificado durante la ejecución del programa (arrays o vectores, matrices..)

El número de elementos de un array se denomina ***rango*** y viene determinado por el valor del índice superior.

Los índices de un array pueden ser constantes, variables o expresiones enteras.

Ejemplo.

i = 0

T[i] representa el elemento que ocupa la posición i .

T[0] es el primer elemento.

T[3] representa el elemento que ocupa la posición 4

T[i + 1] representa el elemento que ocupa la posición 2

La dimensión de un array viene definida por el número de índices que utiliza para hacer referencia a sus elementos. Ejemplo.

matriz [i][j] hace referencia al elemento que ocupa la fila i, y la columna j.

Definimos **Array** como el conjunto de datos del mismo tipo que se almacena en posiciones contiguas de memoria y reciben un nombre común.

Para hacer referencia a un elemento del array se utiliza un índice que indica su posición relativa dentro de la estructura

Pueden ser:

Unidimensionales.- Arrays / Vectores

Bidimensionales.- Matrices .

Multidimensionales.-

Declaración. tipoBase array [ ] = new tipoBase [num] ;

array es una estructura de num elementos de tipoBase (entero, real , cadena…)

num debe ser una variable entera**.**

Ejem.

```
double lluvia [ ] ;  // lluvia es un array de valores de tipo double
```

Aún no hemos definido el número de elementos del array. Igual que para otro tipo de variables, el valor por omisión es null, que representa un array sin ningún valor.

Para indicar el número de elementos, utilizaremos el operador **new** que asigna espacio de memoria al array. lluvia = new double [31];

```
int numDias = 31;
double lluvia [ ] = new double [numDias] ;
```

La variable numDias contiene el número de días del mes.

En este momento disponemos de un array de 31 elementos numerados del 0 al 30 de tipo double, accesible mediante la notación

lluvia [0], lluvia[1] , ... lluvia[30]

Acceso a los componentes

Con cada una de las componentes del array se pueden hacer las mismas operaciones que haríamos con una variable simple de tipo doble.

Ejem.
```
lluvia[3]=30.5;
lluvia[5]=lluvia[3] *2 ;
int i = 6;
lluvia[i+1]=lluvia [i-1];
```

Inicialización.

Existe la posibilidad de inicializar todas las componentes de un array de forma simultánea, o hacerlo individualmente componente a componente.

```
// Individualmente
   double ejem =new double[6];
   ejem[0]=1.3 ;
    ejem[1]=2.678;
   ejem[2]=246.8/2+1;
   ejem[3]=25.7;
   ejem[4]=17.98;
   ejem[5]=12.64 ;
   int i = 6;
   lluvia[i+1]=lluvia [i-1];
```

Simultáneamente

```
double ejem[ ] = {1.3, 2.678, 246.8/2+1, 25.7, 17.98, 12,64 }
```

```
class cre_array {

 public static void main(String[] args) {
  double lluvia[] = new double[31];
  double s=0 ;
  Scanner ent=new Scanner(System.in);
  System.out.println("Dame valores");
   for (int i=0; i<31; i++) {
       lluvia[ i ]=ent.nextInt();
       s=s+lluvia[ i ];
   }
  System.out.println("La media es : "+(s/31));
 } // cierra main()
} // cierra la clase
```

Paso de arrays a métodos.

Los parámetros de tipo array y en general cualquier objeto, se pasan siempre **por referencia.**

En el paso **por referencia** lo que realmente se pasa es la dirección de la variable u objeto, por lo que el papel del parámetro formal es el de ser una referencia al parámetro real.

Las modificaciones que el método pueda hacer sobre los parámetros formales, tienen **efecto directo** sobre los parámetros reales, ya que ambas variables actúan sobre la misma dirección de memoria.

Arrays como parámetros

Para utilizar un array como argumento real en la llamada a un método, se utiliza el nombre de la variable, sin embargo en la definición formal del parámetro es necesario indicar que es un array mediante los corchetes.

```
double media = calcMedia(lluvia);            // llamada al método

static double calcMedia (double lluv [ ] )   // Cabecera de definición del método
```

Ejemplo

Ejemplo.

```
import java.util.*;
class cre_array {
 public static void main(String[] args) {
 double lluvia[] = new double[31];
 double m;
 Scanner ent=new Scanner(System.in);
 System.out.println("Dame valores");
 for(int i=0; i<31;i++)
      lluvia[ i ]= ent.nextInt();
 m=calcMedia(lluvia );
 System.out.println("La media es : "+m);
 } // cierra main()
```

```
static double calcMedia (double lluv[ ] ) {
  double s=0;
  int j=lluv.length;
  for (int c=0; c< j; c++)
      s =+ lluv[c];
  return (s/j);
 }  // cierra metodo calcMedia()


} // cierra clase
```

4.2.- Matrices. Una matriz es un vector de vectores / array de arrays.

Declaración:

```
double matriz [] [] = new double [4] [4];
double matriz [] [] = new double [4] [];
matriz[0]= new double[4];
matriz[1]= new double[4];
matriz[2]= new double[4];
matriz[3]= new double[4];
```

Ejemplo.

```
import java.util.*;
class cre_matriz{
 public static void main(String[] args) {
 double lluvia[] [] = new double[12][31];
 double m;]
 Scanner ent=new Scanner(System.in);
 System.out.println("Dame valores");
 for(m=0; m<12; m++)
  for(int d=0; d<lluvia[m].length; d++)
     lluvia[ m][d]= ent.nextDouble();
  } // cierra main()
}
```

Si se deseara extender el tratamiento del ejemplo anterior, en el tratamiento de la pluviosidad de una zona, para abarcar no sólo los días de un mes sino los de todo un año, se podría definir, por ejemplo, un array de 366 elementos, que mantuviera de forma correlativa los datos de pluviosidad de una zona, día a día.

Con ello, por ejemplo, el dato correspondiente al día 3 de febrero ocuparía la posición 34 del array, mientras que el correspondiente al 2 de julio ocuparía el 184.

Ejemplo. Otra aproximación para la representación de estos datos consistiría en utilizar un array de 12 componentes, arrays a su vez, cada uno de 31 elementos. Lo que permitiría una descripción más ajustada a la realidad y, sobre todo, simplificaría los cálculos de la posición real de cada día en la estructura de datos.

En java las componentes de un array pueden ser, a su vez, otros arrays. Las componentes de estos últimos arrays pueden ser datos elementales, con lo que los arrays son bi-dimensionales (o matrices) o pueden ser arrays de nuevo. En general, no hay límite al anidamiento que puede presentar una estructura de este tipo, con lo que se pueden obtener arrays de tantas dimensiones como se desee (denominados n-dimensionales).

Declaración de arrays multidimensionales.

El código siguiente declara una matriz (array bi-dimensional) de elementos de tipo double, y la inicializa para que sea de 4 por 4 elementos:

```
double matriz [][] = new double[4][4];
```

Es posible inicializar cada uno de los subarrays con un tamaño diferente (aunque el tipo base elemental debe ser **siempre el mismo** para todos los componentes).

Por ejemplo:

```
double matriz [][] = new double [4][];
matriz[0] = new double[3];
matriz[1] = new double[4];
matriz[2] = new double[14];
matriz[3] = new double[10];
```

Acceso a las componentes.

El acceso a las componentes de un array multidimensional se efectúa de forma similar al caso unidimensional, pero tomando en consideración la existencia de múltiples dimensiones. Además, el índice de todos los subarrays se encuentra comprendido entre 0 y la dimensión del mismo menos 1.

Por ejemplo:

```
double matriz [][] = new double[4][4];
for (int i=0; i < 4; i++)
    matriz[i][i] = 1;              // ponemos los elementos de la diagonal principal a 1.
```

Como ocurría con los arrays unidimensionales, a los n-dimensionales se les puede asignar también un valor mediante el uso de un inicializador de array, como por ejemplo, se realiza a continuación:

```
double matriz [][] = { {0,1,2}, {10,11,12}, {20,21,22,23}, {33,11} };
```

Llena una matriz de 4 filas , donde cada fila tiene un número de columnas diferentes.

El atributo length.

Asociado a cualquier variable de tipo array existe un valor (atributo) que determina la longitud real, o número de componentes del array. Dicho atributo se denomina length y se utiliza posponiendo al nombre de la variable la palabra length. Por ejemplo, en el programa anterior ***lluvia.length*** representará el número de componentes del array lluvia (31 elementos). Así, mediante la siguiente instrucción se escribiría dicho valor.

System.out.println(lluvia.length);

4.3 Operaciones con arrays:

Recorrido

- Actualización
- Búsqueda
- Ordenación

Las operaciones con arrays pueden clasificarse en cuatro grandes grupos genéricos:

1.-Recorrido.
2.-Inserción, eliminación , y añadir un elemento al array.
3.-Búsqueda de un elemento, de entre los del array, que cumpla una cierta característica.
4.-Ordenación de los elementos de un array.

La importancia de este tipo de operaciones, proviene de que surgen no sólo en el ámbito de los arrays sino también en muchas otras organizaciones de datos de uso frecuente (como las listas, los ficheros, etc.). Las estrategias básicas de resolución que se verán a continuación son también extrapolables a esos otros ámbitos.

- **Recorrido.** Se clasifican como problemas de recorrido a todos aquellos que para su resolución exigen algún tratamiento de todos los elementos del array.

ejemplo

El orden para el tratamiento de estos elementos puede organizarse de forma ascendente, o descendente. En el **ejemplo** se muestra un método en java para determinar, a partir de un array bidimensional , el día de máxima pluviosidad del mes m;

1 <=m <= lluv.length-1 , así como el día en que ésta se produjo:

```
static void diaMaximo (double lluv[ ][ ], int m) {
int maxD = 1:
for (int i=2; i<lluv[m].length; i++)
        if (lluv[m][i] > lluv[m][maxD] )
                maxD = i;

// maxD es el día de más pluviosidad, al terminar este bucle
System.out.println ("Dia: "+ maxD + " lluvia: " + lluv[m][maxD]);
 }
```

Actualización : Consta de 3 operaciones elementales

***** Añadir

* Insertar

* Borrar

***Añadir*.-** Es la operación de añadir un nuevo elemento al final del vector, (comprobando que haya espacio de memoria suficiente para este nuevo elemento)
***Insertar*.-** Consiste en introducir un nuevo elemento en el interior del vector (es necesario realizar un desplazamiento previo para colocar el nuevo elemento en su posición relativa).
***Borrar*.-** Consiste en eliminar un elemento del interior del vector realizando un desplazamiento para reorganizar el vector.

Ejem. Elimina el elemento que ocupa la posición j en un vector de n elementos

Búsqueda. Básicamente existen dos tipos:

- **Búsqueda Secuencial:**

Recorre secuencialmente el vector desde el primer elemento hasta el último. No se recomienda para vectores de muchos elementos.

- **Búsqueda Binaria o Dicotómica:**

Se utiliza en vectores ordenados, y se basa en la constante división del espacio de búsqueda, comenzando por el elemento que ocupa la posición central del vector.

Se denominan problemas de búsqueda a los que, de alguna manera, conllevan el determinar si existe algún elemento del array que cumpla una propiedad dada. Con respecto a los problemas de recorrido presentan la diferencia de que no es siempre necesario tratar todos los elementos del array, ya que el elemento buscado puede encontrarse inmediatamente, encontrarse tras haber recorrido todo el array, o incluso no encontrarse.

```
class buscaDicot {

  public static void main(String[] args) {
    int v[]={1,34,18,12,6,9,22,61,17,30};
    int  j, busca=17, resp;

// Ordenación previa
   System.out.println("Buscamos el : "+busca);
   resp =busca(v,  busca);
if (resp==-1) System.out.print("\nNo existe el elemento buscado");
    else System.out.print("\nEl elemento está en la posición "+(resp+1));
 }

   static int busca(int b[],  int m){
        int alto=b.length-1, bajo=0, central=0;
        boolean sw=false;

        while (! sw && bajo <= alto) {
           central=(alto+bajo)/2;
            if (b[central] == m) sw=true;
                else if (b[central] > m) alto=--central;
                      else bajo=++central;
          }
          if (sw) return central;
              else return –1;
}
```

Considera el siguiente problema "Determinar, a partir de la matriz lluvia un día de cierto mes que la pluviosidad haya superado cierta cantidad de litros".

El siguiente método, posible solución del problema, determina el día pedido si es que éste existe o señala el hecho de que no exista:

Fíjate que el orden de la evaluación de la condición de la iteración, así como el uso de la conjunción en cortocircuito son relevantes, de ser diferentes el segmento podría originar un error de intento de acceso a un elemento inexistente (fuera de rango).

```
static int superaMedia (double pluv[ ][ ], double lt, int m){
int i=1;
int numDias = pluv[m].length-1;
while ( i<= numDias && pluv[m][i]<=lt ){
 i++;
}
// si se encuentra se devuelve el día; sino se devuelve -1
```

Otra posible solución del problema es utilizar una variable **booleana** que indique en todo momento, si la propiedad buscada ha sido establecida o no.

Por ejemplo, el problema: "Determinar si en el mes de marzo hubo al menos tres días consecutivos con una pluviosidad mayor a 100 litros cada uno de ellos". El siguiente segmento es una de las soluciones posibles del mismo:

Ordenación.

1.- Método de intercambio o burbuja

En numerosas ocasiones nos encontramos con problemas que requieren la ordenación de un conjunto de elementos. El método de la burbuja consiste en comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que estén todos ordenados.

```
Int x;
for (int i= 0; i < v.length; i++) {
  for (int j=i+1; j < v.length; j++) {
    if (v[j-1] > v[j]) {
                        x = v[j-1];
                        v[j-1] = v[j];
                         v[j] = x;
          }
```

1º.- Se comparan v[0] y v[1] si están ordenados se mantienen sino se intercambian entre si.

2º.- Se comparan los elementos v[1] y v[2] de nuevo y si es necesario se intercambian.

3º.- El proceso continua hasta que cada elemento del vector ha sido comparado con el adyacente y se han realizado los cambios necesarios.. La operación para una tabla de n elementos se repetirá n-1 veces.

```
class burbu {
 public static void main(String[] args) {
 int v[]={1,34,18,12,6,9,22,61,17,30};
 int i, j, may;
   System.out.println("Array inicial ");
    for(i=0;i<v.length;i++)
        System.out.print(v[i]+" ");

     for(i=0; i<v.length; i++){
           for(j=1; j<v.length-i; j++)
                 if (v[ j-1] > v[j] ){
                                may = v[j-1] ;
                                  v[j-1] = v[j] ;
                                 v[j]=may;
                            }
  System.out.println("\nRecorrido "+i);
      for(int c=0;c<v.length;c++)
            System.out.print(" "+v[c]);
 }
System.out.println("\nArray ordenado");
  for(i=0;i<v.length;i++)
    System.out.print(v[i]+" ");
 }
}
```

```
class burbu2 {
 public static void main(String[] args) {
   int v[]={1,34,18,12,6,9,22,61,17,30};
   int i, j;
   boolean sw=false;
   System.out.println("Array inicial ");
   for(i=0;i<v.length;i++)
     System.out.print(v[i]+" ");
   for(i=0;i<v.length && !sw;i++)   {
     sw=true;
     for(  j=0; j<v.length-1 ; j++){
          if (v[j] > v[j+1] ) { int may = v[j] ;
                        v[j] = v[j+1] ;
                        v[j+1]=may;
                        sw=false;   }
 }
  System.out.println("\nRecorrido "+i);
 for(int c=0;c<v.length;c++)
    System.out.print(" "+v[c]);
}
   System.out.println("\nArray ordenado");
   for(i=0;i<v.length;i++)
       System.out.print(v[i]+" ");
 }
}
```

2.- Ordenación por selección. Consiste en buscar el elemento menor y colocarlo en la primera posición.

```
Algoritmo selección
Inicio
     Para I =1 hasta n hacer
           Leer a[I]
     Fin_para
// clasificación
     Para i = 1 hasta n - 1 hacer
       Aux = a[i]
       K =I
       Para j = I +1 hasta  n hacer
           Si a[ j ] < aux entonces
                          Aux =a[ j ]
                          K = j
           Fin_si
       Fin_para
       A[ k] = A[I]
       A[I]= aux
   Fin_para

Fin.
```

```
Algoritmo selección
Inicio
// clasificación
      i = 1
     Mientras i < n  hacer
       Aux =A[I]
       K = I
       J = I
       Mientras j < N  hacer
          J =J + 1
          Si A[ j ] < aux entonces Aux =A[ j ]
                                   K = j
          Fin_si
       Fin_mientras
       A[ k] = A[I]
       A[I]= aux
       I = I + 1
     Fin_mientras
Fin.
```

Codifícalos en java