TEMA 5

TIPUS ABSTRACTES DE DADES

Índex de continguts

1 INTRODUCCIÓ

Fins ara havíem treballat amb tipus de dades simples: int, char, double, float, bool... però que passa si necessitarem guardar informació relativa a una persona? Nom, cognoms, DNI, adreça postal, data de naixement etc.. Amb els tipus de dades simples ens seria impossible representar aquesta informació. Es per això que s'utilitzen els tipus de dades compostos que ens permetran agrupar de diverses maneres aquesta informació.

En aquest tema veurem: arrays, cadenes de caràcters, estructures i enumeracions.

Per tal de fer les activitats d'aquest tema es necessitarà de la implementació d'una classe a la que anomenarem TAD que tindrà les següents característiques:

- Classe pública
- Membres de la classe: arrayNúmeros i arrayCaracters
- Mètodes de la classe: recercaBinaria, recercaSequencial, ordenarInserció, ordenarBombolla, ordenarShell, ordernarQuickSort

De moment amb açò serà suficient, en apartats posteriors anirem afegint més membres i mètodes quan ho necessitem.

Activitat 1

Crea la classe 'TAD' amb l'IDE que sols utilitzar basada en les següents especificacions:

- Ha de ser una classe pública
- Ha de tindre mínim una constant que ens indicarà el total d'elements que tindran els arrays, tant el d'enters com el de caràcters.
- Com membres de la classe ha de tindre almenys un array de caràcters i un d'enters. Han de ser privats o públics?
- També ha de tenir un mètode main static on s'iniciarà el programa.

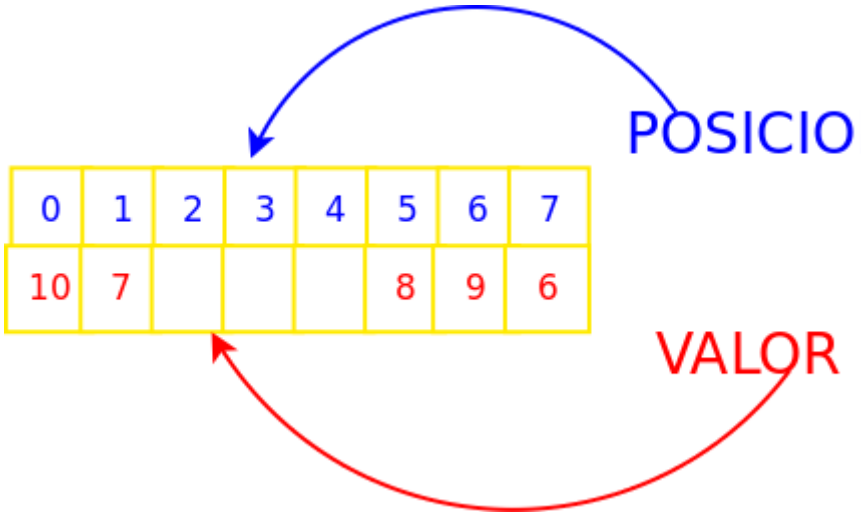
2 ARRAYS

Un array és un tipus de dades compost que permet emmagatzemar un nombre x d'elements del mateix tipus. Amb una única declaració podem tenir accés a un conjunt de valors agrupats. Aquestes agrupacions o arrays poden bé ser de tipus simples o també de tipus compostos.

Per tal d'accedir a cadascun dels elements del array s'utilitza un índex o posició. La primera posició de tot array és la 0 mentre que la segona seria la que té índex 1. Açò pot marejar un poc al principi.

- Array d'enters: {8,2,14,55,3,7}
- Array de caràcters: {'n','a','t','o'}

És molt important distingir entre valor i posició. Mentre que el valor és el contingut de l'array en una posició determinada la posició és l'índex que ens permet recorres l'array i tot el seu contingut.



A l'exemple anterior tenim un array de 8 enters, la part superior de color blau ens indica les posicions que van de la 0 a la 7, 8 posicions ens total. Mentre que a la fila inferior trobarem el valor que conté cadascuna de les posicions. Per exemple en la posició 5 (la sisena) de l'array tindríem emmagatzemat un valor de 8.

Un altre concepte a tenir en compte és la dimensió o grandària de l'array que ens determina el total d'elements que es poden guardar en aquest. En el nostre cas la mida seria de 8.

2.1 DECLARACIÓ I CREACIÓ D'UN ARRAY EN JAVA

Un array en Java és una estructura de dades que ens permet emmagatzemar un conjunt de dades del mateix tipus. El grandària de l'array es determina en la seua declaració i no es permet la seua modificació posterior

```
tipusBase nomArray [];
nomArray = new tipusBase[grandària];
```

Alguns exemples:

```
int mes [];
mes = new int [30];
char grups [];
grups = new char [5];
```

Aquesta forme de declarar i inicialitzar arrays es pot fer en una sola línia de la següent forma:

```
int agendaMensual = new int [30];
```

2.2 INICIALITZACIÓ D'UN ARRAY

Per tal d'accedir a les dades que conté un array necessitem l'identificador d'aquest així com també la posició concreta a la qual volem accedir. Per exemple, imaginem un array d'enters que es diu números.

```
int números. = new int [5]; // Declarem un array
números. [] = {2,4,6,8,10}; // Inicialització de valors
```

D'aquesta forma hauríem creat un array de 5 enters que contindria els nombres parells fins al 10.

Activitat 2

En aquest punt aprofitarem per crear un constructor a la nostra classe TAD. Un constructor de classe és la funció que s'executa en el moment que s'instancia un objecte i és en aquest punt on s'haurien d'inicialitzar aquells elements de la classe que ho necessiten. En el nostre cas els arrays. Inicialitza l'arrayNumeros amb 10 números aleatoris usant la funció rand. Fes el mateix però amb caràcters amb el nostre arrayCaracters. Recorda que els arrays són membres de la classe.

2.3 ACCÉS ALS ELEMENTS D'UN ARRAY

Una vegada ja tenim la declaració de l'array i la seua inicialització, si volem bé accedir al seu contingut o modificar-ho, necessitaríem l'identificador de l'array i l'índex al qual volem accedir. És molt important tenir en compte que el primer element d'un array en Java està a la posició (índex) 0 i no a la posició 1 com ens indicaria la lògica.

```
int primerNumero = números. [0]; // primerNumero valdria 2
números. [0] = 12;  // Canviem el  primer numero de l'array
```

2.4 ARRAYS DE CARÀCTERS

Un array que conté caràcters en lloc de números funciona d'una forma molt semblant al que s'ha explicat en els apartats anteriors. Es pot veure de forma senzilla a l'exemple que vos mostrem a continuació.

```
char vocals = new char [5];  // Declaració de l'array
vocals [] = {'a','e','i','o','u'}  // Inicialització
vocals [0] = 'z'; // Accés als elements de l'array
```

2.5 ARRAYS COM A PARÀMETRES

Com qualsevol altre tipus de dades, un array també es pot passar com a paràmetre a un mètode. Els arrays sempre es passen per referència, és a dir, quan passem un array per paràmetre a un mètode, el que en realitat estem passant és un adreça de memòria que és la que ens indicaria on està l'inici del contingut d'aquest. Passar per referència qualsevol paràmetre significa que qualsevol modificació que es faja dins del mètode afectarà també a l'array fora d'aquest.

No passa el mateix si passem un element concret de l'array, en aquest cas a l'igual que amb els altres paràmetres, aquestos es passen per valor, per tant les modificacions o canvis que es produeixquen dins del mètode no afectaran fora d'aquest.

Activitat 3

Fes dos mètodes per la nostra classe que mostre, de forma amigable, l'array de números i l'array de caràcters. Els mètodes es podrien dir mostraEnters i mostraCaracters i seran privats.

Activitat 4

En la següent activitat afegirem una funció a la nostra classe en la que ens mostre un menú de les accions que podem realitzar. Per tal de realitzar aquest menú afegirem un array d'strings com a nou membre de la classe que inicialitzarem amb les següents opcions:

- «Mostrar array enters»
- «Mostrar array caracters»
- «Reinicialitzar vectors»
- «Recerca seqüencial»
- «Recerca binària»
- «Ordenar amb bombolla»
- «Ordenar amb Quicksort»
- «Ordenar amb Shell»

2.6 BUCLE FOR EN JAVA

Per tal de recórrer arrays podem usar bucles for o while però aquests solen tindre l'inconvenient de que és el programador el que ha de controlar quan s'arriba al final de l'array per tal que l'aplicació no done una errada inesperada i això de vegades no és senzill.

La millor forma de recórrer un array sense haver de controlar quan s'arriba al final és la sentència for usada d'una manera alternativa.

Si per exemple tenim un array d'enters que s'anomena «vector» i volem llistar tots els seus elements, podríem fer el següent:

```
for (int i: vector){
   // Accions a realitzar
}
```

Activitat 5

Fes una funció que mostre per pantalla el menú però esta vegada utilitzant el for com hem explicat a l'exemple anterior.

3 ALGORISMES DE RECERCA

Existeixen dos algorismes de recerca que veurem en aquest apartat: la recerca **seqüencial** i la recerca **binaria**. A l'element que estem buscant li direm **clau**.

3.1 SEQÜENCIAL

També anomenada recerca lineal, es recorren tots els elements de l'array des del principi fins que es troba la clau (element que estem buscant) o fins que s'arriba a l'últim element de l'array.

```
int sequencial (int [] array, int clau) {
   for (int i = 0; i < array.length; i ++ )
        if (array[i] == clau )
             return i;
   return -1;
}
```

3.2 BINARIA

En aquest algoritme existeix la precondició que l'array o vector sobre el qual vaja a fer-se la recerca, ha d'estar ordenat. Aquest mètode és més òptim. El funcionament d'aquest algorisme és el següent: en una primera iteració es compara la clau (element que estem buscant) amb el valor al centre, que s'anomena pivot, de l'array. Poden passar tres coses:

- Que hajam trobat el que busquem, per tant s'acaba la recerca
- Que el pivot siga major que la clau. En aquest cas continuaríem buscant a la meitat esquerra del vector desestimant la meitat dreta (pivot inclòs)
- Que el pivot siga menor que la clau. Amb la qual cosa hauríem de buscar a la part esquerra del vector.

Activitat 6

Afegeix el mètode de recerca seqüencial i el de recerca binaria a la classe TAD. També has d'implementar el codi necessari per tal que quan l'usuari seleccione una de les dues opcions de la nostra aplicació, demane el número a buscar i mostre el resultat per pantalla. En cas que es seleccione la recerca binària, hauríem de comprovar si el vector està ordenat

```
int recercaBinaria (int array[], int clau) {
   int posCentre, posInici, posFinal, valorCentral;
   posInici = 0;
   posFinal = array.length - 1;
   while (posInici <= posFinal ) {
        posCentre = (posInici+posFinal)/2;
        valorCentral=array[posCentre];
        if (clau == valorCentral) {
             return posCentre;
        }
        else if (clau < valorCentral) {
             posFinal = posCentro - 1;
        }
        else {
             posInici = posCentro + 1;
        }
   }
}
```

Activitat 7

Comparem l'eficiència dels dos algorismes de recerca vistos. Fes una funció a la nostra classe TAD en la que es compten el total de vegades que s'accedeix l'array de números per arribar a trobar la clau. Hauràs de posar una opció més al menú, per exemple: «Comparar eficiència dels algoritmes». Aquesta nova opció haurà d'executar varies vegades els dos algorismes de recerca amb vectors de números diferents.

4 ALGORISMES D'ORDENACIÓ

Els algorismes d'ordenació permeten ordenar un array en funció d'un criteri establert al propi algorisme. Hi han diferents tipus d'algorismes d'ordenació: selecció, inserció, bombolla, Shell i Quicksort.

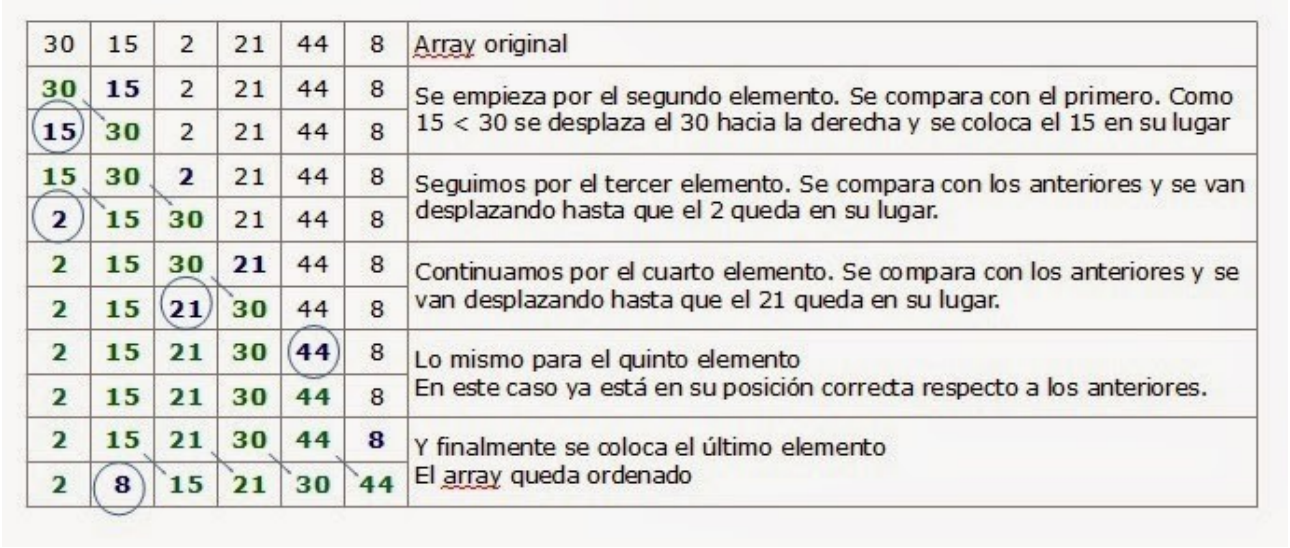
4.1 SEL·LECCIÓ

El procediment d'ordenació per inserció és el següent:

1. Busquem l'element més menut de l'array i el col·loquem a la posició 0 intercanviant els dos elements

2. A partir de la posició 1 busquem l'element més menut següent i fem el mateix que al pas anterior però aquesta vegada a la posició 1 de l'array

3. Repetim el pas 2 fins arribar a l'última posició de l'array.



30	15	2	21	44	8	Array original
30	15	2	21	44	8	Se empieza por el segundo elemento. Se compara con el primero. Como 15 < 30 se desplaza el 30 hacia la derecha y se coloca el 15 en su lugar
15	30	2	21	44	8	
15	30	2	21	44	8	Seguimos por el tercer elemento. Se compara con los anteriores y se van desplazando hasta que el 2 queda en su lugar.
2	15	30	21	44	8	
2	15	30	21	44	8	Continuamos por el cuarto elemento. Se compara con los anteriores y se van desplazando hasta que el 21 queda en su lugar.
2	15	21	30	44	8	
2	15	21	30	44	8	Lo mismo para el quinto elemento En este caso ya está en su posición correcta respecto a los anteriores.
2	15	21	30	44	8	Y finalmente se coloca el último elemento El array queda ordenado
2	8	15	21	30	44	

Activitat 8

Implementa l'algorisme d'ordenació per selecció a la classe TAD i vicula-la a l'opció del menú corresponent.

4.2 INSERCIÓ

Aquest algorisme consisteix en generar un nou array a partir del que ja es té, tal que cada nou element inserit es posa a la seua posició correcta.

Imaginem que tenim el següent array d'enters per ordenar:

{12, 5, 3, 13, 2, 9, 7}

Generem un nou array amb el primer element de l'array a ordenar

{12}

El següent valor de l'array original és 5. És major o menor que el que ja està inserit? Com que la resposta és menor, aquest s'haurà d'inserir abans quedant de la següent forma:

{5, 12}

El nou array es genera ja ordenat. El següent valor de l'array desordenat és 3. On l'haurem de col·locar? Just abans del 5 quedant així:

{3, 5, 12}

Repetim la mateixa operació per cada element de l'array original:

{3, 5, 12, 13}

{2, 3, 5, 12, 13}

{2, 3, 5, 9, 12, 13}

{2, 3, 5, 7, 9, 12, 13}

```
void insercio ( int [] array ) {
   int i, j, aux;
   for (i=1; i < array.legnth; i++ ) {
        j = i;
        aux = array [i];
        while ( j > 0 && aux < array[j-1]) {
             array [j] = array [j-1];
             j--;
        }
   }
   array[j] = aux;
}
```

Activitat 9

Implementa l'algorisme d'ordenació per inserció a la classe TAD i vincula-la a l'opció del menú corresponent.

4.3 BOMBOLLA

L'algorisme d'ordenació per bombolla és el més conegut i també el menys eficient, ja que a cada passada per ordenar un element compara cadascun d'ells amb el seu adjacent de forma que si les seues posicions són incorrectes, s'intercanvien. Al finalitzar cada passada l'element més gran es va posant al final de la llista

Donat el següent array: {12, 5, 3, 13, 2, 9, 7}

Pas 1. Comparem les posicions 0 i 1, és a dir, 12 i 5 i com que estan desordenats (el primer és major que el segon), s'intercanvien.

{5, 12, 3, 13, 2, 9, 7}

Pas 2. A continuació comparem les posicions 1 i 2 de l'array, que són 12 i 3. Com que també estan desordenades, les intercanviem

{5, 3, 12, 13, 2, 9, 7}

Pas 3. Ara anem a comparar les posicions 2 i 3 que contenen els valors 12 i 13 respectivament. Com que estan ben ordenades no es realitza cap acció:

Pas 4. Posicions 3 i 4 que són 13 i 2. Les intercanviem.

{5, 3, 12, 2, 13, 9, 7}

Pas 5. Posicions 4 i 5 que també estan desordenades per tant executem l'intercanviem

{5, 3, 12, 2, 9, 13, 7}

Pas 6. Comparem posicions 5 i 6 que tenen els valors 13 i 7 i s'han d'intercanviar

{5, 3, 13, 2, 9, 7, 13}

Una vegada finalitzada la primera passada hem aconseguit que l'element major de l'array estiga posicionat a l'última posició. A la següent passada recorrerem l'array fins la posició anterior a on es troba ara el número 13, és a dir la posició 5 i aquest procediment es repeteix fins arribar a la posició 0

Finalment el codi quedaria així:

```
void bombolla ( int [] arrayNumeros) {
   for (int i = 0; i < arrayNumeros.length-1; i++) {
        for(int j = i + 1; j < arrayNumeros.length; j++) {
             if (arrayNumeros[i] > arrayNumeros[j]) {
                  int canvi = arrayNumeros[i];
                  arrayNumeros[i] = arrayNumeros[j];
                  arrayNumeros[j] = canvi;
             }
        }
   }
}
```

Activitat 10

Implementa l'algorisme d'ordenació per bombolla a la classe TAD i vincula-la a l'opció del menú corresponent.

4.4 SHELL

L'algorisme Shell és una millora de l'algorisme d'inserció directa. Compara elements separats per un espai de vàries posicions el que permet que un element faja passos més grans fins trobar la seua posició esperada. Els passos múltiples sobre els elements de l'array es fan amb espais de posicions cada vegada més menuts. L'últim pas de l'algorisme Shell és una simple ordenació per inserció però arribats a aquest punt es pot garantir que l'array esta ja quasi ordenat, el que fa que l'ordenació per inserció siga molt eficient.

Vegem com actuaria aquest algorisme aplicat sobre l'array que estem utilitzant com exemple: {12, 5, 3, 13, 2, 9, 7}. Aquest array té un total de 7 elements (n=7). Creem n/2 grups, separats n/2 números entre ells. n/2=7/2=3, és a dir, el nostre interval per crear grups val **3** Marquem en negreta els valors de la primera subllista.

{**12**, 5, 3, **13**, 2, 9, **7**}

Donats els elements 12, 13 i 7 els hem d'ordenar correctament

{**7**, 5, 3, **12**, 2, 9, **13**}

Continuem formant grups de 3 a partir del segon valor de l'array

{7, **5**, 3, 12, **2**, 9, 13}

Ordenem els números en cas de no estar-ho, quedant l'array així:

{7, **2**, 3, 12, **5**, 9, 13}

Passem a l'element 3 de l'array i apliquem el mateix procediment.

{7, 2, **3**, 12, 5, **9**, 13}

Els valors ja estan ordenats en aquest cas, per tant no hem de fer res.

{7, 2, 3, **12**, 5, 9, **13**}

No podem més grups ja que hem arribat al final del vector. Ara el que hem de fer és tornar a dividir el nostre interval entre 2. El nou interval és 3/2 = 1.

```
void shellSort ( int [] array) {
   int interval, i, j, k, temp;
   interval = array.length / 2;
   while ( interval > 0 ) {
        for (i=interval; i<n; i++) {
             j=i-interval;
             while(j>=0) {
                  k=j+interval;
                  if(array[j] <= array[k]
                       j = -1;
                  else {
                       temp = array[j];
                       array[j] = array[k];
                       array[k] = temp;
                       j -= interval;
                  }
             }
        } interval = interval / 2;
   }    }
```

Activitat 11

Implementa l'algorisme d'ordenació shell a la classe TAD i vincula-la a l'opció del menú corresponent.

4.5 QUICKSORT

Aquest és un dels algorismes d'ordenació més ràpids i eficients. Utilitza la tècnica «divideix i venceràs» de tal forma que divideix l'array a ordenar en parts que ordenarà al mateix temps. Bàsicament es tria l'element de l'array com a pivot tal que es forme dos subArrays, en un d'ells s'emmagatzemen els elements menors que l'element que hem triat com a pivot i a l'altre array es guarden els element majors que el pivot. Una vegada fet, s'aplica el mateix procediment als subarrays obtinguts.

Seguint amb el mateix array dels exemples anteriors, aplicarem l'algorisme d'ordenació quicksort.

{12, 5, 3, 13, 2, 9, 7}

Triem un valor que fara el paper de pivot, per exmple el primer element de l'array, el 12 i a partir d'aquest creem dos arrays, un amb els números menors que el pivot i l'altre amb els números majors que el pivot:

pivot: {12}; menors: {5, 3, 2, 9, 7}; majors {13}

El subarray de majors ja està ordenada ja que només té un número. Ara ordenem el subarray de menors: {5, 3, 2, 9, 7} per la qual cosa escollim com a pivot el primer element de l'array, és a dir, el 5.

pivot: {5}; menors {3, 2}; majors: {9, 7}.

Seguim dividint en subarrays, Una vegada els subarrays estan ordenats sempre sempre s'ensamblaran col·locant subarrays de menors, seguides de pivot i subarrays de majors

{2, 3, 5, 7, 9, 12, 13}

El codi quedaria així:

```
void Quicksort ( int [] array, int primer, int ultim) {
   int i, j, central, pivot, tmp;
   central = (primer + ultim)/2;
   pivot = array[central];
   i = primer;
   j = ultim;
   do {
        while (array[i] < pivot)
             i ++;
        while (array[j] > pivot)
             j --;
        if (i <= j) {
             tmp = array[i];
             array[i] = array[j];
             array[j] = tmp;
             i ++;
             j --;
        }
   }
   while (i<=j);
   if (primer < j)
        Quicksort (array, primer, j);
   if ( i < ultim )
        Quicksort (array, i, ultim);
}
```

Activitat 12

Implementa l'algorisme d'ordenació quicksort a la classe TAD i vincula-la a l'opció del menú corresponent.

Activitat 13

Fes que en cada algorisme d'ordenació es mostre una traça on a cada passada de l'algoritme, és a dir, cada vegada que mou un element a l'array, mostre aquest per pantalla.

Activitat 14

Entrega mitjançant la plataforma aules la classe TAD que inclou de les activitats 1 a la 13 amb totes les funcions tant de recerca com d'ordenació sobre arrays unidimensionals.

5 ARRAYS MULTIDIMENSIONALS

Fins ara hem estan utilitzant arrays d'una sola dimensió, coneguts també com vectors, taules o llistes. Aquests arrays es recorren amb ajuda d'un sol índex (tenen una sola dimensió). En aquest apartat estudiarem arrays de més d'una dimensió per als que necessitarem més d'un índex a l'hora de ser recorreguts.

Els arrays multidimensionals més comuns són els de dues dimensions o bidimensionals, també coneguts com matrius. És comú representar aquest tipus de dades com una taula composta per una sèrie de files i columnes:

0,0	0,1	0,2	...	0,n
1,0	1,1	1,2	...	1,n
2,0	2,1	2,2	...	2,n
...	...	...	...	...
n,0	n,1	n,2	...	n,n

Al contrari del que passava amb els arrays d'una sola dimensió on només necessitavem un índex per accedir al valor, en els arrays de dues dimensions com el de l'exemple anterior, necessitarem dos índex per accedir al valor de l'array.

Les matrius, com es coneixen els arrays bidimensionals es forme per files i columnes.

5.1 DECLARACIÓ I CREACIÓ

Per tal de declarar una matriu en java, és a dir, un array de dues dimensions, ho farem de la següent forma:

```
tipusBase matriu [][];
```

Si es fixeu, mentre quan declaravem un array d'una sola dimensió utilitzavem només un parell de claudàtors:

```
tipusBase array[];
```

Ara que volem crear una matriu (array de dues dimensions) utilitzarem dos parells de claudàtors.

És fàcil deduir que si vulguerem declarar un array de tres dimensions, hauriem d'utilitzar tres parells de claudàtors.

Amb aquesta instrucció el que estem fent es declarar la matriu però no li hem dit quina grandària tindrà. Eixa és la diferència entre declaració i creació. Per tal de crear, és a dir, reservar espai per a la nostra matriu, hem de fer el següent:

```
matriu = new tipusBase[n][m];
```

On n i m són valors enters que ens indiquen la grandària de cada dimensió.

Per exemple, per declarar i crear una matriu d'enters de 5 files i 10 columnes fariem el següent:

```
int matriuEnters [][];
matriuEnters = new int [5][10];
// També ho podriem fer en una sola instrucció
int matriuEnters[][] = new int [5][10];
```

5.2 INICIALITZACIÓ I ACCÉS

El procés d'inicialització d'un vector, dona igual les dimensions que tinga, consisteix en donar-li valor a les cel·les que inicialment estan buides. Aquesta inicialització es pot fer de tres formes diferents:

- De forma individual.
- En el moment de la declaració.
- Mitjançant l'ús d'una sentència de control repetitiva.

INICIALITZACIÓ I ACCÉS A CADA ELEMENT DE L'ARRAY

Per accedir a una dada en un array multimensional hem de coneixer els índex de posició d'aquest. En un array de dues dimensions, per accedir a una dada s'ha d'indicar la fila i la columna d'on es troba aquesta informació.

Per exemple, imaginem que tenim la següent taula (array de dos dimensions)

12	13	5	9	22
14	21	11	8	56
23	7	6	10	32
36	24	99	78	55
57	79	18	14	20

Per tal d'inicialitzar-la element a element hauriem d'executar les següents sentències:

```
// matriu[i][j] = valor;
// Inicialitzem la primera fila
matriu[0][0] = 12;
matriu[0][1] = 13;
matriu[0][2] = 5;
matriu[0][3] = 9;
matriu[0][4] = 22;
matriu[0][5] = 14;

// Inicialitzem la segona fila
matriu[1][0] = 21;
matriu[1][1] = 11;
matriu[1][2] = 8;
```

Com es pot observar, inicialitzar una matriu d'aquesta forma pot ser molt pesat, generaria molt de codi i seria molt difícil trobar errades en cas d'haver-les. Imagineu que haguerem d'inicialitzar una matriu de 100 files i 100 columnes així.

INICIALITZACIÓ EN EL MOMENT DE LA CREACIÓ DE LA MATRIU

Una altra forma d'inicialitzar una matriu sense haver d'utilitzar tantes línies de codi és la que es pot observar a l'exemple següent:

```
int [][] mat;
mat = new int [4][5];
mat = {{6,7,5,4,3},{3, 8, 4,1,0}, {1,0,2,1,21}, {9,5,2,25,30}};
```

Però ens trobariem amb el mateix problema a l'inicialitzar una matriu de 100x100, i és que seria molt complicat visualment de seguir el programa.

INICIALITZACIÓ MITJANÇANT L'ÚS DE BUCLES

Si quan treballavem amb arrays d'una dimensió o vectors, utilitzavem un index per recorrer-lo, ara haurem d'afegir un index addicional per cada nova dimensió que tinga el nostre array.

En arrays de dues dimensions el codi necessari per inicialitzar una matriu seria el següent:

```
for (int i = 0; i < FILES; i ) {
   for (int j=0; j< COLUMNES; j++ ) {
        array [i][j] = 0;
   }
}
```

Per tal d'entendre el concepte de matriu o array bidimensional, realitzarem un exercici on treballarem amb les típiques matrius matemàtiques, realitzant les operacions pròpies d'aquestes.

Activitat 15

Es tracta de desenvolupar una aplicació que realitze operacions utilitzant matrius de 3x3. El menú ha de mostrar el següent:

1. Emplena la primera matriu
2. Emplena la segona matriu
3. Visualitza les matrius
4. Suma les matrius
5. Multiplica per un escalar
6. Producte de matrius
7. Transposta
8. Eixir

Per a cadascuna de les operacions anteriors s'ha d'implementar un mètode

Mireu el següent exemple:

Matriu A

1	2
3	4

Matriu B

5	6
7	8

Matriu C= A + B on $C_{ij} = A_{ij} + B_{ij}$

1+5=6	2+6=8
3+7=10	4+8=12

Matriu C = A*B

1*5+2*7=19	1*6+2*8=22
3*5+4*7=43	3*6+4*8=50

Activitat 16

Imagina que has de mantindre les notes dels alumnes de tres assignatures. A cada assignatura tens 15 alumnes, tal que s'ha de mantenir la informació mitjançant una variable que gràficament presenta la següent estructura

Assignatura 1	5	6	7,3	2,3	4,5	…	8,3
Assignatura 2	7,8	8,7	7,7	3,3	4,8	…	6,5
Assignatura 3	7	9	10	8,2	2,8	…	9.2

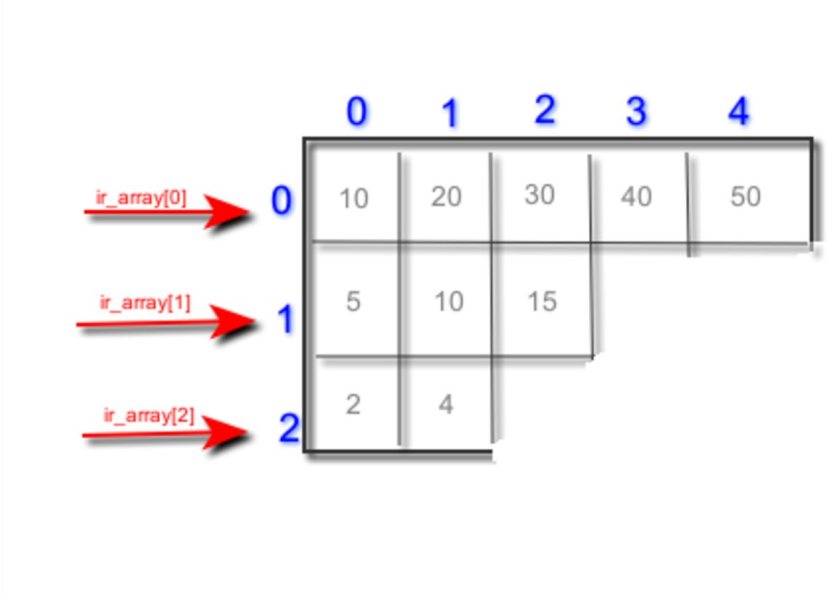
Crea una aplicació en la que pugues:

1. Inserir notes de l'assignatura sel·leccionada.
2. Inserir totes les notes
3. Calcular la nota mitjana de l'assignatura sel·leccionada.
4. Ordenar les assignatures de forma ascendent.
5. Estadística
6. Eixir de l'aplicació.

L'opció estadística mostra la quantitat de notes entre 0 i 3, entre 3,1 i 5, entre 5,1 i 7, entre 7.1 i 9 i entre 9,1 i 10.

5.3 ARRAYS IRREGULARS

Una matriu irregular o escalonada no és més que un array de taules, on cadascuna de les taules que formen l'array no necessàriament han de tenir la mateixa grandària.



La declaració d'un array irregular en java seria de la següent manera:

```
// Declaració d'una matriu (files)
int [][] mat;

// Primer creem la quantitat de files deixant les columnes buit
mat=new int[3][];

// Després creem cada fila indicant la quantitat d'elements
mat[0]=new int[5];
mat[1]=new int[3];
mat[2]=new int[2];
```

6 CADENES DE CARÀCTERS

En Java hem vist que quan volem emmagatzemar un valor enter, definim una variable de tipus int, si pel contrari, el que volem és emmagatzemar un valor amb decimals, definim una variable de tipus double o float. Ara bé, si el que volem és emmagatzemar una cadena de caracters, per exemple el nom d'una pesona, hem de definir un objecte de tipus String

```
// Crear un objecte string
String strNom = «Manolo el del bombo»;
```

Aquest codi el que fa és crear un objectes string strNom que conté el nom: «Manolo el del bombo». També podriem crear un string de la següent manera:

```
String strNome = new String(«Manolo el del bombo»);
```

Al tractar-se d'una classe, la forma natural de treballar amb ella serà mitjançant l'ús dels mètodes que disposa la classe. Aquests mètodes són:

- int length(): retorna la llargària de la cadena en un enter.
- char charAt (int i): ens diu quin caracter està a la posició 'i'
- String substring(int i): ens retorna la subcadena que hi ha a partir de la posició 'i' fins el final de la cadena
- String substring(int i, int j): ens retorna la subcadena que es troba des de l'índex i fins el j
- String concat(String str): concadena la cadena 'str' que es passa com a paràmetre al final de la cadena. Per exemple:

```
String s1 = «Java»;
String s2 = «Pego»;
String s3 = s1.concat(s2); // El resultat seria «JavaPego»
```

- int indexOf(String s): Ens retorna l'index dins de la cadena de la primera aparició de la subcadena s. Per exemple:

```
String s1 = «Java a Pego»;
int pos = s1.indexOf(«Pego»); // Retorna 7
```

- int indexOf(String s, int i): retorna l'índex dins de la cadena de la primera aparició de la subcadena s a partir de l'índex i
- int lastIndexOf(int ch): torna l'índex de l'última vegada que apareix el caracter 'ch' dins de la cadena.
- boolean equals(Sring str): Compara l'string amb l'objecte que es passa per paràmetre.
- boolean equalsIgnoreCase (String otroString): Compara dues cadenes sense tenir en compte majúscules i minúscules.
- int compareTo (String otroString): compara dues cadenes lexicogràficament. En altres paraules, diu quina és major que l'altra.
- int compareToIgnoreCase (String otroString): com el mètode anterior però sense tenir en compte majúscules ni minúscules.
- String toLowerCase(): converteix la cadena a minúscules.
- String toUpperCase(): converteix la cadena a majúscules.
- String trim(): suprimeix els espais en blanc que puguen haver als extremps de la cadena
- String replace (char oldChar, char newChar): substitueix totes les ocurrències de oldChar que hi ha a la cadena per newChar

7 ENUMERACIONS

Una enumeració és un tipus especial de 'classe' que representa un grup de constants. Cada element d'aquesta estructura està associada a un valor de un tipus de dades concret (normalment enter) on el primer element de l'enumeració sol agafar el valor 0

```
// Exemple 1. Dies de la setmana
enum Dies {
   Dilluns,
   Dimarts,
   Dimecres,
   Dijous,
   Divendres,
   Dissabte,
   Diumenge
}

// Exemple 2. Talles de roba
enum Talles {
   XXL,
   XL,
   L,
   M,
   S,
   XS
}
```