UD7.- Estructures de Dades I

Mòdul: Programació
1r DAM
Curs 2018-2019

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

CONTINGUTS

- Tipus de dades
- Strings
- Arrays

Tipus de dades en Java

- Tipus de dades primitius
  - Numèrics
    - Enters: byte, int, short, long
    - Reals: float, double
  - Caràcter: char
  - Lògics: boolean

- Tipus de dades referència
  - String, Array, Class (objectes), Interface

Tipus primitius *versus* tipus referència

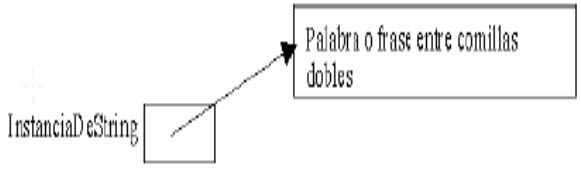
- **Variables de tipus primitius**
  - Contenen directament les dades.
  - Cada variable té la seua pròpia còpia de les dades, de forma que les operacions en una variable d'un tipus primitiu no poden afectar a una altra variable.

- **Variables de tipus referència**
  - Contenen una referència o punter al valor de l'objecte, no el propi valor.
  - Dos variables de tipus referència poden referir-se al mateix objecte, de forma que les operacions en una variable de tipus referència poden afectar a l'objecte referenciat per un altra variable de tipus referència.

Tipus de dades *String*

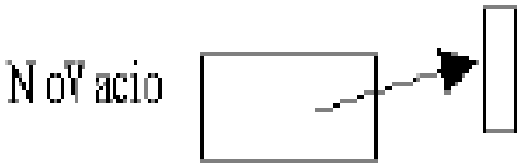
- Per a Java una cadena de caracters no forma part dels tipus primitius sinó que és un objecte de la classe *String* (java.lang.String)
- Els objectes de la classe *String* es poden crear:
  - Implícitament: s'ha de posar una cadena entre cometes dobles. Per exemple, a l'escriure `System.out.println("Hola")`, Java crea un objecte de la classe *String* automàticament.
  - Explícitament:
    - `String str=new String("Hola");` // o també
    - `String str="Hola";` // mode tradicional

      Encara que sembla el mateix, la forma de crear-se l'objecte no és igual. La segona forma sempre crea un nou objecte en el heap, una zona de memòria especial per a les variables dinàmiques, mentre que la primera forma pot crear o no un nou objecte (si no el crea el reutilitza del String Pool, una memòria caché dissenyada per a reciclar Strings).



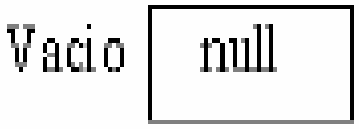
Declarar i inicialitzar un *String*

- Per a crear un *String* sense caràcters es pot fer:
  - `String str=””; //o`
  - `String str=new String();`



- És un cadena sense caràcters però és un objecte de la classe *String*.

- Però si fem:
  - `String str; //o`
  - `String str=null;`



- S'està declarant un objecte de la classe *String*, però no s'ha creat l'objecte (el seu valor és null).

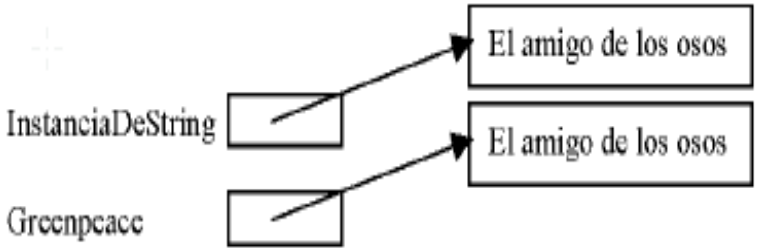
Alguns mètodes de la classe *String* (I)

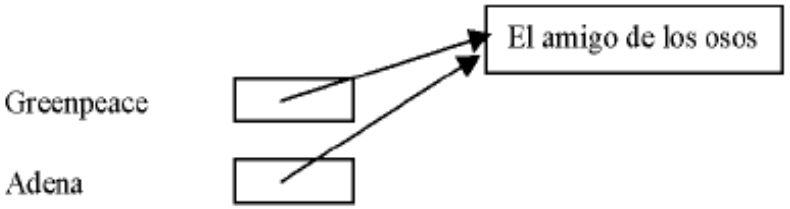
- Per utilitzar-los, RECORDA:
  - Si el mètode és dinàmic
    - variableString.mètode(arguments)
  - Si el mètode és estàtic
    - String.mètode(arguments)
  - La posició del primer caràcter d'un string és 0, no 1.

- Davant de qualsevol dubte, CONSULTA l'API de Java, disponible a :
  - http://docs.oracle.com/javase/7/docs/api/

Comparar objectes *String* (I)

- És important tindre en compte que si tenim dos instàncies de la classe *String* que apunten a continguts idèntics, això no significa que siguen iguals aplicant l'operador comparació (==).





- Dos strings seran iguals (==) si apunten a la mateixa estructura de dades (adreça de memòria).
- Per tant, els objectes *String* no poden comparar-se directament amb els operadors de comparació (==).

Comparar objectes *String* (II)

- Disposem del següents mètodes:
  - **cadena1.equals(cadena2)**
    - *true* si la cadena1 és igual a la cadena2.
    - Les dos cadenes són variables de tipo *String*.
  - **cadena1.equalsIgnoreCase(cadena2)**
    - Igual que l'anterior, però no es tenen en compte majúscules i minúscules.

Comparar objectes *String* (III)

- **cadena1.compareTo(cadena2)**
  - Compara les dos cadenes considerant l'ordre alfabètic (de la taula ASCII)
  - si la primera cadena major que la segon: retorna 1
  - si són siguals: retorna 0
  - si la segon és major que la primera: retorna -1

- **cadena.compareToIgnoreCase(cadena2)**
  - Igual que l'anterior, però ignorant majúscules

Alguns mètodes de la classe *String*

- **length**
  - Retorna la longitud d'una cadena
    - `String text="prova2";`
    - `System.out.println(text.length());` //escriu 6

- Per a concatenar cadenes es pot fer de dos formes: amb el mètode **concat** o amb l'operador **+**.
  - `String s1="Bon", s2="dia", s3, s4;`
  - `s3=s1+s2;`
  - `s4=s1.concat(s2);`

Alguns mètodes de la classe *String*

- **charAt**
  - Retorna un caràcter de la cadena.
  - El caràcter que s'ha de retornar s'indica per la seua posició.
  - Si la posició és negativa o sobrepassa la longitud de la cadena, es produix un error d'execució.
    - `String s1=”prova”;`
    - `char c1=s1.charAt(2);` //c1 valdrà o

Alguns mètodes de la classe *String*

- **substring**
  - Retorna un troç d'una cadena.
  - El troç va des d'una posició inicial fins a una posició final (la posició final no s'inclou).
  - Si les posicions indicades no són valides, es genera una excepció.
    - `String s1=”Bona vesprada”;`
    - `String s2=s1.substring(5,10);` //s2=vespr

Alguns mètodes de la classe *String*

- **indexOf**
  - Retorna la primera posició en la qual apareix un text concret en la cadena.
  - En cas de que no es trobe la cadena buscada, retorna -1.
  - El text a buscar pot ser *char* o *String*.
    - `String s1="Quería decirte que quiero que te vayas";`
    - `System.out.println(s1.indexOf("que"));` // escriu 15
  - Es pot buscar des d'una posició determinada. Seguint l'exemple anterior:
    - `System.out.println(s1.indexOf("que",16));`
      `// ara escriu 26`

Alguns mètodes de la classe *String*

- **lastIndexOf**
  - Retorna la última posició en la qual apareix un text concret en la cadena.
  - És quasi idèntica a indexOf, però comença a buscar des del final.
    - `String s1="Quería decirte que quiero que te vayas";`
    - `System.out.println(s1.lastIndexOf("que"));` // Dona 26
  - També permet començar a buscar des d'una determinada posició.

Alguns mètodes de la classe *String*

- **endsWith**
  - Retorna *true* si la cadena acaba amb un text en concret.
    - `String s1=”Quería decirte que quiero que te vayas”;`
    - `System.out.println(s1.endsWith(“vayas”));`//dona true

- **startsWith**
  - Retorna true si la cadena comença amb un text en concret.

Alguns mètodes de la classe *String*

- **replace**
  - Canvia totes les aparicions d'un caràcter per un altre en el text que s'indica i ho emmagatzema com a resultat.
  - El text original no es canvia, per la qual cosa s'ha d'assignar el resultat de replace a un *String* per guardar el text canviat:
    - `String s1=”cosa”;`
    - `System.out.println(s1.replace('o','a'));` // escriu casa
    - `System.out.println(s1);` // continua amb cosa

Alguns mètodes de la classe *String*

- **replaceAll**
  - Modifica en un text cada entrada d'una cadena per una altra i retorna el resultat.
  - El primer paràmetre és el text que es busca (que pot ser una expressió regular) i el segon paràmetre és el text amb el qual es reemplaça el text buscat. La cadena original no es modifica:
    - `String s1="Cazar armadillos";`
    - `System.out.println(s1.replaceAll("ar","er"));` // escriu Cazer ermadillos
    - `System.out.println(s1);` // escriu Cazar armadillos

Alguns mètodes de la classe *String*

- **toUpperCase**
  - Retorna la versió en majúscules de la cadena.
- **toLowerCase**
  - Retorna la versió en minúscules de la cadena.
- **toCharArray**
  - Obté un array de caracters a partir d'una cadena.

Alguns mètodes de la classe *String*

- **String.valueOf** (mètode estàtic)
  - Este mètode no és exclusiu de la classe *String*, també està present en altres classes (per example Integer) i sempre convertix valors d'una classe a una altra.
  - En el cas dels objectes *String*, permet convertir valors que no són de cadena a forma de cadena.
    - Exemples:
      - `String numero = String.valueOf(1234);`
      - `String data = String.valueOf(new GregorianCalendar());`
  - En els exemples s'observa que és un mètode estàtic.

Classe *StringBuilder*

- La classe *String* representa una cadena de caracters no modificable (Inmutable)
  - Una operació com convertir a majúscules no modificarà l'objecte original sinó que retornarà un nou objecte amb la cadena que resulte de l'operació.
- La classe *StringBuilder* representa una cadena de caracters modificable tant en contingut com en longitud.
- NOTA: La classe *StringBuffer* té la mateixa funcionalitat (constructors i mètodes) però *StringBuilder* té major rendiment.

Alguns mètodes per a *StringBuilder*

- Mètodes **length** i **capacity**
  - Retornen, respectivament, la quantitat real de caracters que conté l'objecte i la quantitat de caracters que l'objecte pot contindre.
- Método **append**
  - Afegim caracters al final de l'objecte.
- Mètodes **delete**, **replace** i **insert**
  - Modifiquen l'objecte actual

Classe *StringBuilder*

- Hem vist que podem concatenar cadenes de caràcters fent ús de l'operador **+** i també mitjançant el mètode **concat** de la classe String.
- Ara bé, els String són “inmutables” i per tant sols es poden crear llegir però no es poden modificar.
- Examinem el següent codi:

```
public String getMensaje(String[] palabras){
   String mensaje="";
   for(int i=0; i<palabras.length; i++){
      mensaje+=" "+palabras[i];
   }
   return mensaje;
}
```

Classe *StringBuilder*

- Cada vegada que s'afig una nova paraula, es reserva una nova porció de memòria i es rebutja la vella porció de memòria que és més xicoteta (una paraula menys) perquè siga alliberada pel recol·lector de fem (garbage collector). Si el bucle s'executa 1000 vegades, hi haurà 1000 porcions de memòria que el recol·lector de fem ha d'identificar i alliberar.
- Per a evitar aquest treball extra al recol·lector de fem, es pot emprar la classe StringBuffer que ens permet crear objectes dinàmics, que poden modificar-se.

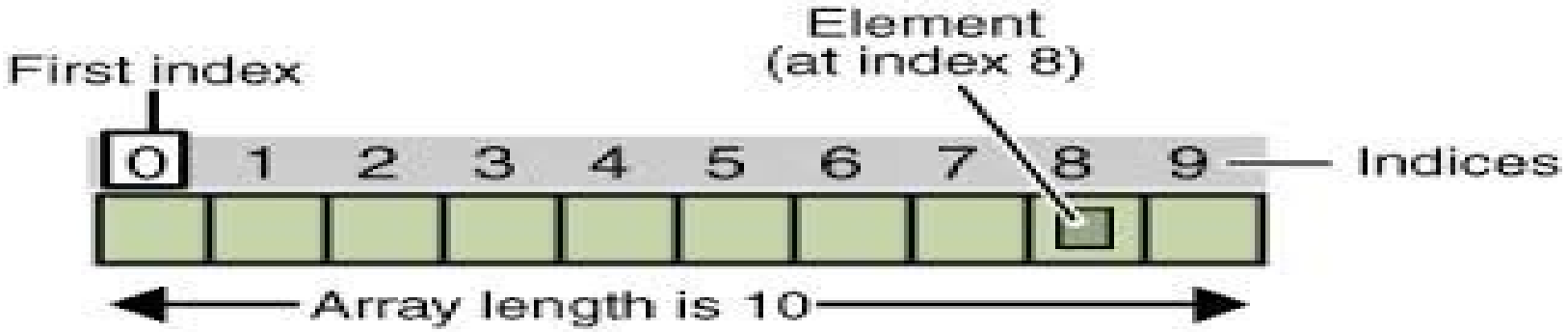
```
public String getMensaje(String[] palabras){
   StringBuilder mensaje=new StringBuilder();
   for(int i=0; i<palabras.length; i++){
      mensaje.append(" ");
      mensaje.append(palabras[i]);
   }
   return mensaje.toString();
}
```

Què és un *array*? (I)

- Un array és una **sèrie d'elements**.
  - Tots els elements de l'array tenen el mateix tipus de dades.
  - S'accedix a cada element mitjançant un índex enter.
  - La primera posició de l'array és 0.
- La longitud de l'array es determina al crear-lo.
  - El nombre d'elements no pot ser modificat en temps d'execució. Els arrays són estàtics. No podrem ni eliminar posicions ni insertar posicions.
- Al crear un array, si no s'especifiquen valors, totes les posicions són inicialitzades al valor per defecte del tipus de dades de què es tracte.

Què és un *array*? (II)

- Un array en Java és un classe especial (definida en java.util.Arrays).
- Un array pot contindre tipus primitius o tipus complexes.
- Si intentem accedir a una posició fora de l'array es produirà un error i es llançarà una excepció (java.lang.IndexOutOfBoundsException)



Declarar i crear un *array* unidimensional (I)

- És diferent...
  - Declarar una variable array
    - Es crea una variable capaç d'apuntar (contindre l'adreça de memòria) a un objecte array.
    - L'objecte encara no està creat i la variable apunta o conté null.
  - Crear o instanciar un array
    - Creem físicament l'objecte. Se li assignen posicions de memòria.
    - S'utilitza la paraula reservada *new* i s'invoca al constructor.
  - Podem declarar i instanciar en la mateixa instrucció.

Declarar i crear un *array* unidimensional (II)

- Declarar:
  - tipus_dades[] nomArray;
  - Exemple: int[] numeros;
- Crear un array i assignar-lo:
  - nomArray = new tipus_dades[num_elements]
  - Exemple: numeros=new int[10];
- Declarar + crear un array
  - tipus_dades[] nomArray = new tipus_dades[num_elements]
  - Exemple: int[] numeros = new int[10];
- Declarar + crear + inicialitzar:
  - tipus_dades[] nomArray = {v1, v2, v3, ...};
  - Exemple: int[] numeros={53, 15, -22, 60, 6, 8 ,14, -75,12, 64};

Accés als elements d'un *array* unidimensional

- Si tenim: int[ ] numeros = {2, -4, 15, -25};
- Si un array unidimensional **a** té n elements:
  - Al primer element s'accedix amb a[0]
  - I a l'últim element s'accedix amb a[n-1]

- Accés per a lectura (extraure el valor):
  - Exemple: System.out.println(numeros[3]);

- Accés per a escriptura (posar un valor):
  - numeros[2] = 99;

Recorregut dels elements d'un *array* unidimensional (I)

- És una operació molt freqüent recòrrer els elements d'un array.
- Es pot utilitzar un bucle amb comptador...

```
int[] datos = {1, 2, 3, 4};
for(int i=0; i<datos.length; i++){
  System.out.println(datos[i] + " ");
}
```

- O iterar sobre els elements...

```
for(int dato : datos){
  System.out.println(dato + " ");
}
```

Recorregut dels elements d'un *array* unidimensional (II)

- Exemple:

```
int maximo = Integer.MIN_VALUE;
for (int i = 0; i < vector.length; i++) {
   if (vector[i] > maximo)
   maximo = vector[i];
}
```

```
int maximo = Integer.MIN_VALUE;
for (int n: vector) {
   if (n > maximo)
   maximo = n;
}
```

Grandària dels arrays. Arrays vs. llistes

- Els *arrays* són de mida fixa, mentre que les llistes són de mida variable.
- Si no sabem la grandària d'un *array* al crear-lo, tenim dos possibilitats:
  - 1.- Crear un *array* molt gran, de forma que càpiguen les dades. Mala gestió de l'espai (es malgasta).
  - 2.- Crear un *array* de grandària reduïda, però tindre en compte que si arriben més dades s'haurà d'ampliar (és a dir, crear un *array* més gran i copiar les dades). Es malgasta temps d'execució.
- Les llistes (classe List) són de mida variable. Les llistes són una forma còmoda d'aplicar la segona opció. Ho veurem més endavant...

Arrays i mètodes

- Podem passar un *array* com a paràmetre a un mètode, tenint en compte que **els canvis que realitzem sobre l'*array* en el procediment invocat es mantindran** al tornar el flux de l'execució al procediment invocador.

- Això és degut a que **els *arrays* són de tipus referència** i, per tant, les variables de l'*array* amb les quals treballem tant des del procediment invocador com des del procediment invocat, són en realitat punters cap a una mateixa zona de memòria o referència (la que conté l'*array)*.

Arrays i mètodes. Exemple

- Per tant, quan es crida a un mètode i se li passa un *array*, el mètode fa la seua còpia de la referència, però compartix l'*array*.

```
void caso1(int[] x) {
   x[0] *= 10;
}

void test1() {
   int[] a = {1, 2, 3};
   System.out.println(Arrays.toString(a));
   caso1(a);
   System.out.println(Arrays.toString(a));
}
```

Execució

[1, 2,3]
[10, 2, 3]

Còpia d'*arrays* (I)

- Quan una variable de tipus *array* es fa igual a una altra, es còpia la referència (i es **compartix** l'*array*):

```
void copia1() {
    int[] a = {1, 2, 3};

    System.out.println(Arrays.toString(a));
    int[] b = a;
    System.out.println(Arrays.toString(b));
    a[0] *= 10;
    System.out.println(Arrays.toString(a));
    System.out.println(Arrays.toString(b));
}
```

Execució
[1, 2, 3]
[1, 2, 3]
[10, 2, 3]
[10, 2, 3]

Còpia d'*arrays* (II)

- Si a més de compartir la referència volem una còpia de l'*array*, es pot emprar el mètode **clone()** :
  - Si els elements de l'*array* són d'un tipus **primitiu**, es copia el seu valor.

```
void copia2() {
    int[] a = {1, 2, 3};
    System.out.println(Arrays.toString(a));
    int[] b = a.clone();
    System.out.println(Arrays.toString(b));
    a[0] *= 10;
    System.out.println(Arrays.toString(a));
    System.out.println(Arrays.toString(b));
}
```

Execució
[1, 2, 3]
[1, 2, 3]
[10, 2, 3]
[1, 2, 3]

Còpia d'*arrays* (III)

- Amb el mètode **clone()**, si els elements de l'*array* són objectes, es copia la referència (es compartix l'objecte).

```
void copia2Objetos() {
    Punto[] a = {new Punto(1, 2), new Punto(3, 4)};
    System.out.println(Arrays.toString(a));
    Punto[] b = a.clone();
    System.out.println(Arrays.toString(b));
    a[0].multiplica(-1);
    System.out.println(Arrays.toString(b));
}
```

Execució

[(1,2), (3,4)]
[(1,2), (3,4)]
[(-1,-2), (3,4)]

Còpia d'*arrays* (IV)

- Per últim, la còpia es pot fer de forma explícita.

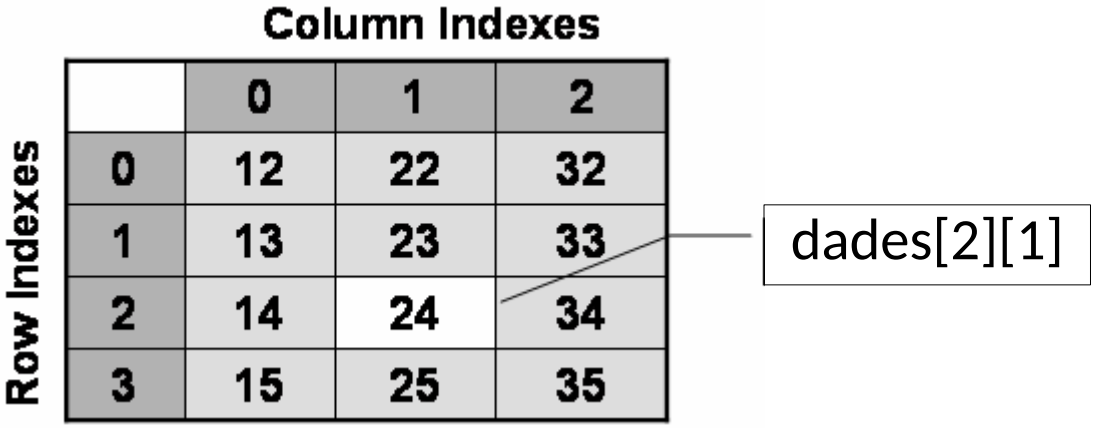
```
tipo[] a = ...;

tipo[] b = new tipo[a.length];

for (int i = 0; i < a.length; i++) {

    b[i] = a[i];
}
```

Alguns mètodes dels *arrays*

- Propietat pública **length:** retorna el nombre d'elements de l'*array*.
- **Arrays.sort():** ordena un *array*.
- **Arrays.binarySearch():** busca un valor en un *array*. L'*array* ha d'estar ordenat.
- **Arrays.equals():** compara arrays.
- **Arrays.fill():** assigna valors a un array.
- **Arrays.toString():** convertix un *array* en *String*.
- **Arrays.copyOf(), clone()**: copien un *array*.

Arrays multidimensionals o matrius

- Podem pensar en una matriu de 2 dimensions com si fora una quadrícula.



- Declarar una matriu de 4 files i 3 columnes de números enters:
  - `int[][] dades = new int[4][3];`
- Assignar un valor a un element específic de la matriu:
  - `dades[2][1] = 24;`

Declarar i crear un *array* multidimensional

- Declarar:
  - tipus_dades[][] nomArray;
  - Exemple: int[][] numeros;
- Crear un array i assignar-lo:
  - nomArray = new tipus_dades[num_files][num_columnes]
  - Exemple: numeros=new int[10][5];
- Declarar + crear un array
  - tipus_dades[][] nomArray = new tipus_dades[num_elements]
  - Exemple: int[][] numeros = new int[10][5];
- Declarar + crear + inicialitzar:
  - tipus_dades[][] nomArray = {v1, v2, v3, ...};
  - Exemple: int[][] numeros={{1, 2, 3}, {4, 5, 6}};

Arrays bidimensionals o matrius

- En realitat, un *array* multidimensional és un *array* d'*arrays*.
- Per exemple:
  - Un *array* 4x3 és un *array* de 4 elements, en el qual cada un d'ells és un *array* de 3 elements:
    - dades[0] és un *array* de 3 elements
    - ...
    - dades[3] és un *array* de 3 elements
- Podem tindre *arrays* bidimensionals no quadrats (cada fila pot tindre un nombre diferent de columnes)
  - int[ ][ ] numeros = new int[4][ ];
  - numeros[0]=new int [7];
  - ...
  - numeros[3]=new int[3];

Arrays bidimensionals o matrius

- length
  - Proporciona el nombre d'elements o longitud de l'*array*
  - matriu.length //nombre de files
  - matriu[0].length //nombre de columnes de la primera fila
  - matriu[1].length //nombre de columnes de la segona fila

Recorregut dels elements d'una matriu

- Es pot emprar un bucle amb comptador

```
double[][] matriz={{1,2,3,4},{5,6},{7,8,9,10,11,12},{13}};

for(int i=0; i < matriz.length; i++) {
    for(int j=0; j < matriz[i].length; j++) {
        System.out.print(matriz[i][j]+"\t");
    }
    System.out.println("");
}
```

- o iterar sobre els seus elements

```
for(double[] fila : matriz) {
    for(double dato : fila) {
        System.out.print(dato + " ");
    }
    System.out.println("");
}
```

Arrays multidimensionals. Exemple

```
public class MatrizUnidadApp {
   public static void main (String[] args) {
      double[][] mUnidad= new double[4][4];
      for(int i=0; i < mUnidad.length; i++) {
         for(int j=0; j < mUnidad[i].length; j++) {
            if(i == j) {
               mUnidad[i][j]=1.0;
            }
            else {
               mUnidad[i][j] = 0.0;
            }
         }
      }
      for(int i=0; i < mUnidad.length; i++) {
         for(int j=0; j < mUnidad[i].length; j++) {
            System.out.print(mUnidad[i][j]+"\t");
         }
         System.out.println("");
      }
   }
}
```

Arrays multidimensionals

- Per a crear una matriu multidimensional:
  - tipusDades[ ][ ][ ]... nomVariable = new tipusDades[dimensió1] [dimensió2][dimensió3]...

- total elements = dimensió1 x dimensió x dimensió3 ...