Estructuras de Almacenamiento:

ARRAYS

¿Qué es un array?

- *Es una estructura de datos que almacena una cantidad fija de elementos del mismo tipo, a los cuales se puede acceder por medio de un índice que indica su posición dentro de la estructura.*
- Es una estructura homogénea de datos
  →Todos los datos son de un mismo tipo o clase.
- Estructura estática de datos:
  →Los *arrays contienen siempre el mismo número de datos*
- En álgebra: vectores (1D), matrices (2D).
- En Java: Son objetos que se declaran, crean, inicializan y consultan.

Arrays en Java

- En Java, los Array pueden ser de cualquier tipo de dato, incluidos objetos.
- El tipo de dato Array es, a su vez, un objeto.
- Las variables del tipo Array se declaran utilizando **[]**, del siguiente modo:
  tipo_basico[] nombre_variable;

3

Declaración y creación de arrays

Modificador_acceso tipo nombre[];

Después de la declaración hay que crearlos con new:

nombre=new tipo[tamaño];

Al crear el array se reserva espacio en memoria para sus componentes. No se puede modificar posteriormente su tamaño.

Se puede declarar y crear simultáneamente:

Ej**. int array[]=new int[10];**

String nombres[]=new String[10];

4

Inicialización

- Una vez creado el array, todas sus posiciones son inicializadas al valor por defecto, sin son atributos miembro. Dependiendo del tipo de dato declarado: (numérico 0, objetos null, false para boolean). Si son variables locales no se inicializan.
- Inicialización explícita:

```
int array[]= {1,23,5}; /*se crea con
esos tres componentes*/
```

5

Longitud de un array. El atributo length

Los arrays son objetos. Los objetos tienen atributos que pueden consultarse si son públicos.
Se puede preguntar al array cuál es su longitud.
Java define **el atributo length** que guarda el número de elementos del *array:*
nombre_array.length
Ejemplo:

```
int serie[] = new int[10];
serie.length // nos devuelve el valor 10
```

6

Uso de arrays: Indexación de elementos

Nos podemos referir a cualquier elemento del array mediante el nombre del array seguido de la posición del elemento entre corchetes.
El número de posición se denomina subíndice.
Consideremos:

```
int [] serie = new int[10];
for (int i = 0; i < 10; i++)
    serie[i] = i * i;
```

Los elementos del *array ocupan posiciones de memoria contiguas,* que se numeran empezando por cero:

- El primer elemento es **serie[0] y el último serie[9].**
- Para referirnos al iésimo elemento escribimos **serie[i-1].**

7

Indexación de elementos

Los elementos de un *array se comportan como cualquier otra variable de su* tipo y pueden usarse en expresiones.
Ejemplo:

```
int suma = serie[0] + serie[1];
```

Los subíndices pueden ser cualquier expresión entera que evalúe un valor dentro de los límites del *array.*
Ejemplo:

```
int a = 4;
int b = 3;
serie[a + b] += 2; // sumar dos a serie[7]
```

8

Longitud de un *array: IndexOutOfBoundsException*

Consideremos el ejemplo:

```
int a = 4;
int b = 3;
serie[a + b] += 2;
```

*¿Qué ocurre si **serie se ha** declarado así?*

```
int serie[] = new int[5];
```

O así:

```
int serie[] = new int[7];
```

- *No debemos acceder a posiciones del array inexistentes.*
  - En el mejor de los casos estamos accediendo a algo indefinido.
  - En el peor podemos destruir información útil.
- La JVM detecta los intentos de acceder a posiciones no existentes y lanza una excepción que provoca (si no es tratada) que el programa aborte: **IndexOutOfBoundsException**

9

Bucle for-each

Esta estructura nos permite recorrer una Colección o un array de elementos de una forma sencilla. Evitando el uso de Iteradores o de un bucle for normal.

La estructura del bucle for-each sería de la siguiente forma:

```
for (TipoBase variable: ArrayDeTiposBase) {..}
```

Ejemplo:

```
String array[] = {"Avila", "Burgos", "León", "Palencia",
  "Salamanca", "Segovia", "Soria", "Valladolid", "Zamora"};
for (String elemento: array)
  System.out.println(elemento);
```

10

Recorrido

No se puede sobrepasar los límites del array, si esto ocurriera se crearía una excepción del tipo: ArrayIndexOutOfBoundException

```
String arrayCadenas[]=new String[5];
for (int i=0;i<arrayCadenas.length;i++)
        {
        tec=new Scanner(System.in);
        System.out.println("Introduce un texto ");
        arrayCadenas[i]=tec.nextLine();
    }
System.out.println("Las cadenas leidas son:");
 for (String elem : arrayCadenas)
    System.out.println(elem);
```

11

Observaciones

No hay restricciones respecto del tipo de datos que puede contener un array.
→ Pueden ser tanto tipos de datos primitivos como referencias a objetos de una clase.
• Los arrays tienen un tamaño fijo:
→ No pueden crecer ni disminuir su tamaño.
→ Se dice que son estructuras estáticas de datos (en contraposición con las estructuras dinámicas de datos que si pueden variar el tamaño).
• No obstante, las referencias de arrays se pueden asignar a arrays de diferentes tamaños.

12

Búsqueda en *arrays:* Búsquedas lineal y binaria

Se trata de determinar si el array contiene un elemento que coincide con el valor de una clave.
→ La clave de búsqueda.
Se denomina búsqueda al proceso de encontrar ese elemento clave.

- En Arrays desordenados se realiza la búsqueda:
  - lineal o secuencial

- En Arrays ordenados se realiza la búsqueda :
  - lineal o secuencial ordenada
  - binaria o dicotómica

13

ARRAYS DESORDENADOS

Búsqueda en *arrays*: Búsqueda lineal

Búsqueda lineal: *compara cada elemento del array con la clave de búsqueda.*

- Si el *array* no está ordenado es igualmente probable que el elemento buscado esté al principio o al final.
  → El número medio de comparaciones es igual a la mitad del tamaño del array.
- La búsqueda lineal funciona bien en *arrays* pequeños, pero es muy poco eficiente en *arrays* grandes y en *arrays* ordenados.
- Orden de complejidad: `O(n)`

```
// Definimos como variables de instancia
int a[] = {2,3,6,1,4,7,9,4,3,5};
int clave = 7;

public static void main (String args []) {
 for ( int n = 0; n < a.length; n++ )
  if (a[n] == clave)
   {System.out.println("Lo encontré en "+ n);
    break;}
}
```

14

Búsqueda lineal 2

```
int v[]={2,8,6,3,1,5,6,3,9};
int clave=6;
while (i< v.length-1 && v[i]!=clave)
    i++;
 if (v[i]==clave)
    System.out.println("\n el valor esta en la posicion:
  "+i);
  else
   System.out.println("\n El valor buscado NO existe");
```

15

Ejemplo1 Búsqueda Lineal

```
//Una sola ocurrencia
boolean enc=false;
int pos;
for (int i=0;i<arrayCadenas.length && !enc;i++)
        {
        if (arrayCadenas[i].equals("pepe"))
        { enc=true;
          pos=i;
         }
        }
     if (enc)
        System.out.println("Enc en posicion "+pos);
  else
    System.out.println("NO Enc");
```

16

Ejemplo2 Búsqueda Lineal

```
//Varias ocurrencias
boolean enc=false;
for (int i=0;i<arrayCadenas.length;i++)
        {
        if (arrayCadenas[i].equals("pepe"))
        { enc=true;
         System.out.println("Enc en posicion "+i);
         }
        }
      if (!enc)
        System.out.println("NO Enc");
```

17

ARRAYS ORDENADOS

Búsqueda en *arrays*: Búsqueda binaria

Algoritmo de búsqueda binaria.

Comparar la clave con el elemento situado en la mitad del array (elemento medio).

- **IF** *(clave == elemento medio) → Hemos terminado.*
- **ELSE IF** *(clave < elemento medio) → buscar en la primera mitad del array,*
- **ELSE** *→ buscar en la segunda mitad del array.*

El proceso se repite sucesivamente hasta que:

- *La clave sea igual al elemento medio del subarray que queda.*

OR

- *El subarray que queda consta de un solo elemento distinto de la clave*

18

Búsqueda en *arrays*: Búsqueda binaria

Es un algoritmo muy eficiente.

El algoritmo de búsqueda binaria elimina de la búsqueda la mitad de los elementos que quedan en el array después de cada comparación.

1024 elementos ($1024 = 2^{10}$) → 10 comparaciones como máximo.
1.048.576 elementos (2^{20}) → 20 comparaciones como máximo

El número máximo de comparaciones es el exponente de la primera potencia de 2 mayor que el número de elementos del array.

Orden de complejidad: $O(\log_2 n)$

Pero sólo funciona con arrays ordenados

19

Búsqueda en *arrays*: Búsqueda binaria

```
int [] a = {2,4,6,8,10,12,14,16,18,20,22}
```
Clave = 20

Subíndice	0	1	2	3	4	5	6	7	8	9	10
Valor	2	4	6	8	10	12	14	16	18	20	22

2			
4			
6			
8			
10			
12			
14	14		
16	16		
18	18		
20	20	20	20
22	22	22	

20

Búsqueda binaria

```
class BusquedaBinaria {
public static void main (String args []) {
int a[ ] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22};
int inf = 0;
int elem=20;
int sup = a.length -1;
int medio;
while (inf <= sup) {
  medio = (inf + sup) / 2;
  if (elem == a[medio]) //se encontró
   {System.out.println("Lo encontré en " + medio);
    break;}
  else
    if (elem < a[medio])
      sup = medio-1;
    else
      inf = medio+1;}}}
```

21

Búsqueda secuencial ordenada

```
int v[]={2,4,7,9,34};
int clave=8;
int i=0;
   while(i<v.length-1 && v[i]<clave ){
          i++;
   }
   if(v[i]==clave)
     System.out.println("Encontrado posicion :"+i);
   else
     System.out.println("No existe el valor buscado");
```

22

Ordenación de *arrays*: Algoritmo de la burbuja.

Array ordenado: Un array a está ordenado si se cumple que:

∀ *pareja de subíndices* `i,j, si i ≤ j ⇒ a[i] ≤ a[j]`

Algoritmo de la burbuja:

Sea el *array* `a`. Repetir (`a.length - 1`) veces:

1. *Realizar una pasada por el array.*
2. *Comparar pares de elementos sucesivos.*
   - → *Si un par está en orden se deja como está.*
   - → *Si no se intercambian los valores.*

En el *array* compara en cada pasada `a[0]` con `a[1]` , `a[1]` con `a[2]` , `a[2]` con `a[3]` y así sucesivamente.

Ordenación de arrays: Algoritmo de la burbuja.

- Un valor grande puede bajar varias posiciones de una sola pasada, pero uno pequeño sólo puede subir una posición.
- En la primera pasada el valor más grande quedará en la última posición del *array* (`a.lenght -1`).
- En la segunda pasada el segundo valor más grande quedará en la penúltima posición.
- Se puede comprobar que hacen falta un número de pasadas igual a la longitud del *array* menos 1.

```
int a[] = {4, 3, 2, 1}
```

Inicial	Pasada 1	Pasada 2	Pasada 3
4	3	2	1
3	2	1	2
2	1	3	3
1	4	4	4

Pasada 1	Al entrar	Al salir
i = 0 →	4, 3, 2, 1	3, 4, 2, 1
i = 1 →	3, 4, 2, 1	3, 2, 4, 1
i = 2 →	3, 2, 4, 1	3, 2, 1, 4

Ordenación de *arrays: Algoritmo de la burbuja.*

```
public class Ordenacion{

    // Suponemos que está inicializado
    public static void main(String args[] ) {
        int [] a = {1,3,-2,3,45,2,98,-45,0};
        int intercambio;
        for (int i = 0; i < a.length-1; i++ ){
            for (int j = 0 ; j < a.length- 1- i; j++){
                if (a[j] > a[j+1]){
                  intercambio = a[j];
                   a[j] = a[j+1];
                   a[j+1] = intercambio;
                }
            }
        }
        for (int elem : a)
          System.out.print(elem+" ");
    }
}
```

25

ArraysMultidimensionales en Java

Java no maneja directamente arrays de múltiples subíndices, pero como los elementos de un array pueden ser cualquier cosa →
Java permite especificar arrays de un subíndice cuyos elementos son también arrays de un subíndice.
⇒ El resultado en la práctica es que se pueden manejar tablas de dos o más subíndices

26

Arrays Bidimensionales (I)

Las tablas que requieren dos subíndices para identificar un elemento se llaman *arrays* con doble subíndice.

- *El primer subíndice se refiere a la fila (es el índice del array de arrays)*
- *El segundo subíndice se refiere a la columna (es el índice de cada uno de los arrays del array de arrays)*
- *Un array de dos dimensiones de m filas y n columnas es un array* `m x n`.

Ejemplo: En la tabla se muestra la disposición de los elementos de un *array* a de `3x4`:

```
int [][] a = new int[3][4];
```

	Columna 0	Columna 1	Columna 2	Columna 3
Fila 0 `a[0]`	`a[0][0]`	`a[0][1]`	`a[0][2]`	`a[0][3]`
Fila 0 `a[1]`	`a[1][0]`	`a[1][1]`	`a[1][2]`	`a[1][3]`
Fila 2 `a[2]`	`a[2][0]`	`a[2][1]`	`a[2][2]`	`a[2][3]`

`a` es un *array* de `3x4` enteros.

`a[i]` es un *array* de `4` enteros.

`a[i][j]` es un entero

27

Arrays Bidimensionales (II)

Los *arrays* de dos dimensiones (en general de cualquier dimensión) se pueden declarar e inicializar de manera similar a los *arrays* de una dimensión.

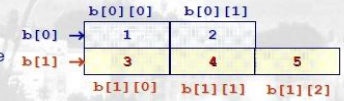
Ejemplo: *Array* con doble subíndice

```
int b [][] = { {1,2}, {3,4,5} }
```

Los valores **se agrupan por fila** encerrados entre llaves:

`1` y `2` inicializan `b[0][0]` y `b[0][1]`

`3`, `4` y `5` inicializan `b[1][0]`, `b[1][1]` y `b[1][2]`



- En la declaración aparecen dos parejas de corchetes, porque Java mantiene *arrays* de *arrays*. Hablar de *arrays* de dos o más dimensiones no es estrictamente correcto.
- Lo que estamos declarando es un *array* cuyos elementos son dos *arrays*, uno conteniendo dos elementos `{1,2}` y el otro conteniendo tres elementos `{3,4,5}`.

28

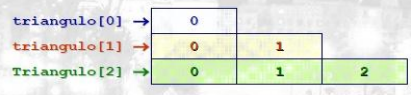
Arrays Bidimensionales (III)

```
int [][] matriz = { {11,12,13}, {21,22,23} };
System.out.println(matriz [0][2]);   // Imprime en pantalla 13
```

```
int [][] triangulo = new int [3][];
triangulo[0] = new int[1];
triangulo[1] = new int[2];
triangulo[2] = new int[3];

triangulo[0][0] = 0;
triangulo[1][0] = 0;
triangulo[1][1] = 1;
triangulo[2][0] = 0;
triangulo[2][1] = 1;
triangulo[2][2] = 2;      O bien,

int[][]  triangulo = { {0} ,{0,1} ,{0,1,2} };
```



¡ Triangulo tiene dos dimensiones pero no es una matriz !

Arrays Bidimensionales (IV)

Suma de los elementos de un *array* bidimensional:

```
class Recorrido_Array {

   int a[][] = { {1,2}, {4} };
   int total = 0;

   public static void main (String args[] ) {

      for (int fila = 0; fila < a.length;  fila++)
         for (int col = 0; col < a[fila].length; col++)
            total += a[fila][col];

   }
}
// a.length da el número de filas (número de elementos del array a)
// a[fila].length da el número de columnas (número de elementos
// del array a[fila])
```

Arrays Bidimensionales (V)

Se trata de calcular la nota máxima y mínima de un *array* de calificaciones, donde cada fila representa un estudiante y cada columna representa una calificación en diferentes exámenes

```
class Calificación {
   int notas[][] = {{77,68,86,73}, {96,87,89,81}, {70,90,86,81}};

   public static void main (String args[]){
      int nota_baja = 100, nota_alta = 0;
      for (int i = 0; i < notas.length; i++ )
         for (int j = 0; j < notas[i].length; j++ )
            if (notas[i][j] < nota_baja) nota_baja = notas[i][j];
      System.out.println("La nota baja es" + nota_baja);

      for (int i = 0; i < notas.length; i++)
         for (int j = 0; j < notas[i].length; j++ )
            if (notas[i][j] > nota_alta) nota_alta = notas[i][j];
      System.out.println("La nota alta es" + nota_alta);
   }
}
```

31

Arrays bidimensionales

- Se puede organizar la información en tablas de dos dimensiones, con dos índices para acceder a los elementos, uno representa las filas y el otro las columnas.
- Con la instrucción:

  int[][] mat = new int[10][10]

Se crea una matriz capaz de almacenar 100 elementos del tipo int.

32

Declaración e inicialización de una matriz:

```
int matrizEnteros = new int [3][7];
for (int i =0; i < matrizEnteros.length;i++)
{
   for (int j=0;j < matrizEnteros[i].length;j++)
   {
        matrizEnteros[i][j] = 1;
   }
}
```

33

Declaración e Inicialización

```
public class Matriz
{ public static void main(String[] args)
{ int[][] cat ={ { 1,2,3}, { 4,5,6,7,8,9,10},
   {11,12}}; //filas de longitud variable
for(int i= 0;i<cat.length;i++)
{ for (int j=0;j<cat[i].length;j++)
   System.out.print(cat[i][j]+ " ");
System.out.println();}}}
```

34

Matrices de Longitud Variable

```
int num[][]=new int[5][]; //Hay 5 filas
  num[0]=new int[10]; //Primer array
  num[1]=new int[20]; //Segundo array
  num[2]=new int[30]; //...
  num[3]=new int[20];
  num[4]=new int[10];
```

35

```
public class Mat{
 public static void main(String[] args){
 int[][] a = new int[3][];//array de 3 arrays de
   enteros
a[0] = new int[3];
a[1] = new int[2];
a[2] = new int[1];
a[0][0]= a[0][1]= a[0][2]=11;
a[1][0] = 21; a[1][1] = 22; a[2][0] = 31;
 for(int i= 0;i<a.length;i++) {
for (int j=0;j<a[i].length;j++)
   System.out.print(a[i][j]+ " ");
   System.out.println(); } }}
```

36

Matrices recorrido

```
for (int i=0;i<num.length;i++)
      for (int j=0;j<num[i].length;j++)
          num[i][j]=(int)(Math.random()*10+i);
System.out.println("La matriz generada es:");
   for (int i=0;i<num.length;i++)
          {
             System.out.println("\nFila "+i);
             for (int j=0;j<num[i].length;j++)
                System.out.print(num[i][j]+" ");
          }
```

37

Ordenación de *matrices: Algoritmo de la burbuja.*

```
public static void ordenar(int m[][]){
    int sw=0;
    int aux;
    while (sw==0){
       sw=1;
       for(int f=0;f<m.length-1;f++){
          int h=0;
          while((m[f][h]==m[f+1][h])&& h<m[f].length-1){
             h++;
          }
          if(m[f][h]>m[f+1][h] && h<m[f].length-1){
             for(int c=h; c<=m[f].length-1;c++){
                aux=m[f][c];
                m[f][c]=m[f+1][c];
                m[f+1][c]=aux;
             }
             sw=0;
          }
       } }
}
```

38