Arrays

Objetivos

1) Declarar e inicializar arrays básicos. Tamaño. Salida por consola y reinicializar. Acceso a las celdas de un array
2) Recorrer total y parcialmente un array en todos los sentidos
3) Arrays como objetos: copiar, clonar y comparar
4) Métodos con arrays y librerías
5) Parámetros por referencia de tipos básicos
6) Procesamiento de múltiples arrays
7) Arrays dinámicos, de caracteres y cadenas
8) Ordenar arrays

Tablas

Objetivos

1) Declarar e inicializar tablas. Reinicializar. Tamaño. Acceso a las celdas de una tabla. Mostrar una tabla por consola
2) Recorrer total y parcialmente una tabla en todos los sentidos
3) Recorrer tablas dentadas
4) Copiar y clonar una tabla
5) Métodos con tablas. Librerías.

Clases envoltorio

Objetivos

1) Conocer la tipos envoltorio asociado a los tipos básicos. Rango y tamaños de los tipos envoltorios
2) Declaración e inicialización de tipos envoltorios. Entrada y salida
3) Conversión de tipos básicos y su correspondiente tipo envoltorio
4) Expresiones de tipos envoltorio. Conversión entre tipos envoltorio. Conversión de tipos envoltorio a tipos básicos
5) Conversión entre tipos envoltorio y cadenas
6) Conversiones a binario, octal y hexadecimal. Máximo y mínimo. Infinite y NaN
7) Ordenar arrays de tipo envoltorio
8) Métodos con arrays

Colecciones

Objetivos

1) Saber qué es una colección
2) Manejo de ArrayList
3) Manejo de HashSet
4) Manejo de Stack
5) Manejo de Hashtable

Arrays

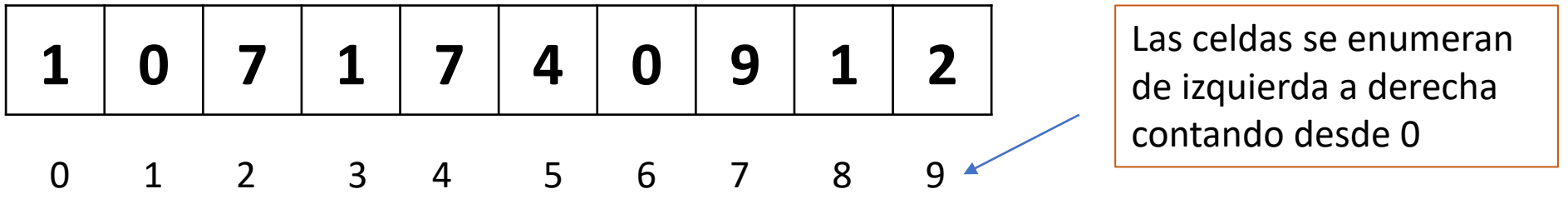
Objetivos

1) Declarar e inicializar arrays básicos. Tamaño. Salida por consola y reinicializar. Acceso a las celdas de un array
2) Recorrer total y parcialmente un array en todos los sentidos
3) Arrays como objetos: copiar, clonar y comparar
4) Métodos con arrays y librerías
5) Parámetros por referencia de tipos básicos
6) Procesamiento de múltiples arrays
7) Arrays dinámicos, de caracteres y cadenas
8) Ordenar arrays

Arrays básicos

Los arrays son una secuencia de variables que se almacenan en memoria una a continuación de la otra.

Un ejemplo de un array con 10 celdas enteras sería:



Los **arrays básicos** son arrays cuyas celdas contienen valores de un tipo básico (boolean, char, byte, short, int, long, float, double) o de cadena (String)

Declaración e inicialización de arrays

Ejemplos de declaración e inicialización de arrays básicos serían:

```
//Variables
int[] a = {23,0,2,3};
double[] b = new double[5];
float[] c = null;
char[] v = = {'a','e','u'};
String[] cad = {"Luis","Ana","Juan"};
String[] cad = new String[2];
```

23	0	2	3

Se utiliza cuando se sabe el tamaño del array y el valor inicial de cada celda

Se utiliza cuando sólo se sabe el tamaño del array

0.0F	0.0F	0.0F	0.0F	0.0F

Como valor inicial de cada celda se pone para los tipos byte, short, int, long, float y double el 0; para el tipo char, '\0'; para el tipo boolean "false" y para el tipo String, el null

Se utiliza cuando no se sabe el tamaño del array. El array no existe y no es accesible

'a'	'e'	'u'

null	null

"Luis"	"Ana"	"Juan"

Declaración e inicialización de arrays (II)

Se pueden declarar e inicializar varías variables del mismo tipo de array:

```
//Variables
int[] a = {0,1,2}, b = null, c = new int[5];
```

Se pueden declarar e inicializar a la vez variables de un tipo y variable de array del mismo tipo:

```
//Variables
double n1 = 0, a[] = {1,0,2.3}, b[] = null, n2 = 1.3;
```

Arrays de tamaño 0

Es posible definir arrays con 0 elementos del siguiente modo:

```
//Variables
long[] a = {}, b = new long[0];
boolean c[] = {}, d[] = new boolean[0];
```

Los arrays de tamaño 0 existen, no tienen celdas pero son accesibles
Los arrays con el valor null no existen y no son accesibles

```
//Variables
char[] a = null;
```

?

Salida por consola de arrays

```
//Variables
int[] a = {23,4,50};
double[] b = {56.6,0.001};
char[] c = {'a','z','?'};
boolean[] d = {true,false,false};
String e[] =
{"España","EEUU","Rusia"};
int[] f = {};
int[] g = null;

//Programa
System.out.println( Arrays.toString(a) );

System.out.println(Arrays.toString(b));
System.out.println(Arrays.toString(c));
System.out.println(Arrays.toString(d));
System.out.println(Arrays.toString(e));
System.out.println(Arrays.toString(f));
System.out.println(Arrays.toString(g));
```

Se puede utilizar el método “toString” de la clase “Arrays” del paquete “java.útil” para mostrar por consola un array

Convierte el array “a” en una cadena → Obtiene una cadena con los valores de “a” separados por comas y entre corchetes

Salida consola

```
[23, 4, 50]
[56.6, 0.001]
[a, z, ?]
[true, false, false]
[España, EEUU, Rusia]
[]
null
```

Tamaño de un array

A partir del nombre de un array se puede obtener su tamaño siempre y cuando el array exista (sea distinto de null)

```
//Variables
int[] a = {200,3,4};
int[] b = new int[0];
int[] c = null;

//Programa
System.out.println( a.length );

System.out.println( b.length );

System.out.println( c.length );
```

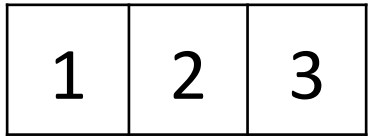
```
3
0
Exception in thread "main"
java.lang.NullPointerException:
Cannot read the array length
because "c" is null
```

a.length devuelve un valor entero que representa el número de celdas que tiene el array “a”

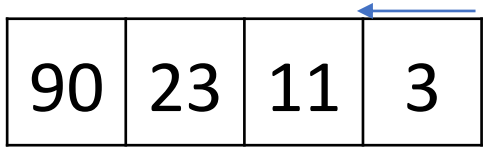
Reinicialización de un array

A lo largo de la ejecución de un programa se puede reinicializar un array del siguiente modo:

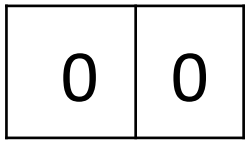
```
//Variables
int[] a = {1,2,3};
int n = 2;
//Programa
a = new int[]{90,23,11,3};
a = new int[n];
```



"a" se incializa con los valores 1,2 y 3



A partir de ahora "a" tiene 4 celdas con lo valores 90, 23, 11 y 3

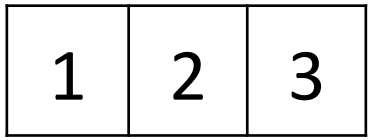


"a" se reinicializa a 0,0

Observa que como nuevo tamaño se puede poner una expresión entera no negativa

Constantes de array

Una constante de array es un array que una vez se ha establecido su tamaño, ya no se puede modificar.




```
//Constantes y variables
final int[] a = {1,2,3};

final double[] b;
int n = 0;

//Programa
n = in.leerInt("Tamaño: ",v->v>0);

b = new double[n];
```

"a" siempre tendrá 3 celdas

La constante "b" se inicializa en tiempo de ejecución

"b" tendrá tantas celdas como el valor de "n". "b" ya no se puede reinicializar

El operador []

El acceso al valor almacenado en una celda de un array se realiza con el operador [].

```
//"a" int[]
```

0	1	2	3
23	0	7	1

```
//"c" String[]
```

0	1	2
“Luis”	“Ana”	“Juan”

```
a[1] → 0
a[5]
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException:
Index 5 out of bounds for length 4
```

Se puede poner como valor entre los corchetes cualquier expresión cuyo resultado sea un entero

```
//Variables
int i = 2;
a[i] → 7
c[i-1] → "Ana"
a[n.length-1] → 1
```

Última celda de “a”

Instrucciones con acceso a una celda

Ejemplos de instrucciones sobre el array “a”

```
//Variables
int i = 0;
int[] a = {23,0,7,1};
```

0	1	2	3
23	0	7	1

`a[0]-=10;`

13	0	7	1

`a[i]=a[i+2]+1;`

8	0	7	1

`if(a[i]>0) a[i]++;`

24	0	7	1

`a[2*(i+1)]=a[2*i+1);`

23	0	0	1

`a[a.length-1]--;`

23	0	7	0

`a[(int) (Math.random()*a.length)] = 9;`

23	9	7	1

Se genera un número aleatorio entre 0 y 3. Por ejemplo el **1**

Ejemplo de **instrucciones con acceso a celdas**

Sea un array de reales inicializado con los valores 2.5, 1.7 y 0.8. Muéstrese los valores en una misma línea separados por un espacio. Duplíquese el tercer valor y calcúlese la media de los tres valores redondeada a un decimal

```
//Variables
double[] a = {2.5,1.7,0.8};
double media = 0;

//Programa
System.out.println(a[0]+" "+a[1]+" "+a[2]);
a[2]=a[2]*2;
media = (a[0]+a[1]+a[2])/3;
media = (int) (media*10+0.5)/10.0;
System.out.println(media);
```

```
2.5 1.7 0.8
1.9
```

Salida consola

Recorrer un array

Uso de for

```
int[] a = {23,0,7,1};
```

0	1	2	3
23	0	7	1

Recorrer el array de izquierda a iderecha

Rellenar aleatoriamente

3	2	9	0

```
for(int i=0;i<a.length;i++)
   a[i] = (int)(Math.random()*10);
```

Incrementar cada celda en 1

24	1	8	2

```
for(int i=0;i<a.length;i++)
   a[i]++;
```

Recorrer el array de derecha a izquierda

Contar pares

23	0	7	1

```
int con = 0;
for(int i=a.length-1;i>=0;i--)
   if(a[i]%2==0) con++;
```

Obtener el mayor

23	0	7	1

```
int mayor = a[0];
for(int i=a.length-1;i>=0;i--)
   if(a[i]>mayor) mayor=a[i];
```

Media

23	0	7	1

```
int tot = 0; double media = 0;
for(int i=0;i<a.length;i++)
   tot+=a[i];
media = (double) tot/a.length;
```

Mostrar

23	0	7	1

```
for(int i=0;i<a.length;i++)
{
   if(i>0) System.out.print(", ");
   System.out.print(a[i]);
}
```

Salida consola

```
23, 0, 7, 1
```

Recorrer un array

Hágase un programa que rellene un array de enteros con valores aleatorios entre 0 y 9, ambos inclusive, cuyo tamaño positivo se introduce por consola. El programa mostrará los datos separaos por comas, obtendrá la media y contará los pares.

```
//Constantes y variables
final int N;
int[] a = {};
int con = 0, total = 0;
//Programa
N = in.leerInt("Tamaño: ",v->v>0);
a = new int[N];

for(int i=0;i<a.length;i++)
{
    a[i] = (int) (Math.random()*10);
    total+=a[i];
    if(a[i]%2==0) con++;
    if(i>0) System.out.print(", ");
    System.out.print(a[i]);
}
System.out.println();
System.out.printf("Media: %1.1f\n",(double)
total/a.length);
System.out.println("Pares: " + con);
```

```
Tamaño: 6
6, 6, 9, 3, 6, 2
Media: 5,3
Pares: 4
```

Salida consola

No se utiliza una variable de cadena para la salida

Todos los procesos (rellenar, acumular, contabilizar y mostrar) se hacen con un solo “for”

Recorrer un array (II)

Resolver el ejercicio anterior utilizando varios “for”

```
//Constantes y variables
final int N;
int[] a = {};
int con = 0, total = 0;
String s = "";

//Programa
N = in.leerInt("Tamaño: ",v->v>0);
a = new int[N];

/* Rellenar array */
for(int i=0;i<a.length;i++)
   a[i] = (int) (Math.random()*10);

/* Acumular valores */
for(int i=0;i<a.length;i++)
   total+=a[i];

/* Contabilizar pares */
for(int i=0;i<a.length;i++)
   if(a[i]%2==0) con++;

/* Salida datos array */
for(int i=0;i<a.length;i++)
{
   if(i>0) s+=", ";
   s+=a[i];
}
s+="\n";

/* Salida media */
s+= String.format("Media:
%1.1f\n",(double) total/a.length);

/* Salida pares */
s+= "Pares: " + con;

/* Mostrar salida */
System.out.println(s);
```

Cada proceso se hace en un “for”

Se utiliza una variable de cadena para la salida

Salida consola

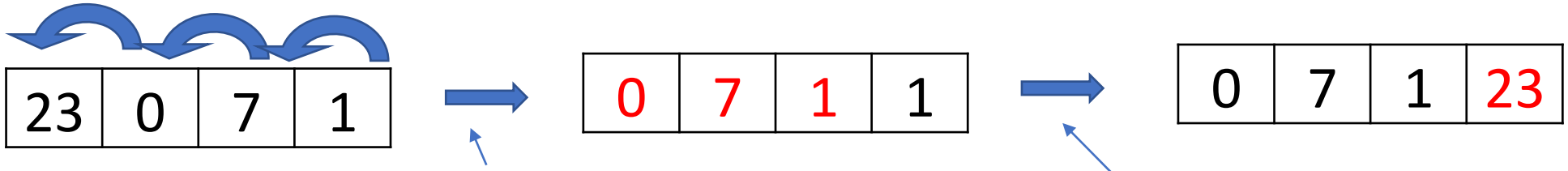
```
Tamaño: 6
6, 6, 9, 3, 6, 2
Media: 5,3
Pares: 4
```

Recorrer parcialmente un array

Uso de for `int[] a = {23,0,7,1};`

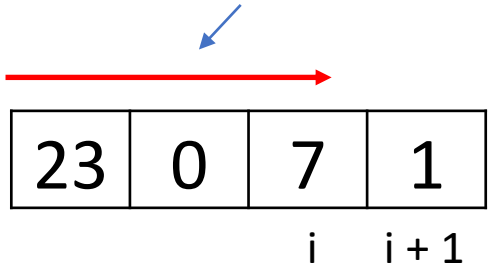
0	1	2	3
23	0	7	1

Mover hacia la izquierda los datos, dejando en el último el primero



Mover los tres últimos datos a los tres primeros
a[0] = a[1], a[1] = a[2], a[2] = a[3] → a[i] = a[i+1]
desde i =0 hasta i=2 (a.length-2)

En el último se pondrá el primero
a[a.length-1] = aux (en aux se almacena a[0] para no perderlo)



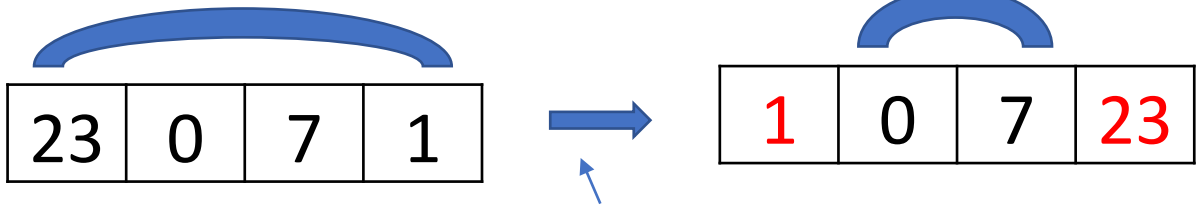

```
int aux = 0;
aux = a[0];
for(int i=0;i<a.length-2;i++)
    a[i] = a[i+1];
a[a.length-1] = aux;
```

Recorrer parcialmente un array

Uso de for `int[] a = {23,0,7,1};`

0	1	2	3
23	0	7	1

Invertir el array

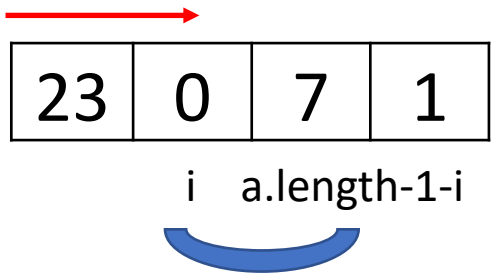


Intercambiar a[0] con a[3]

Intercambiar a[1] con a[2]

Observa que la suma de los índices de los elementos a intercambiar siempre vale **3** (**a.length-1**) . Por tanto el valor en la posición i se tiene que intercambiar con el de la posición **3**-i, es decir, **a.length-1**-i

Observa que el proceso de intercambiar sólo se hace hasta la mitad (**a.length/2**)




```
int aux = 0;
for(int i=0;i<a.length/2;i++)
{
    aux = a[i];
    a[i] = a[a.length-1-i];
    a[a.length-1-i] = aux;
}
```

Se recorre la mitad de los elementos

Se intercambian los valores de las posiciones “i” y “a.length-1-i”

Recorrer un array con saltos

Uso de for `int[] a = {23,0,7,1};`

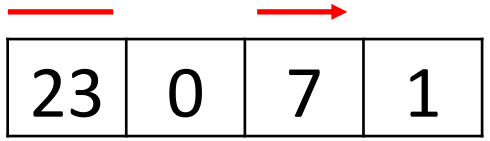
0	1	2	3
23	0	7	1

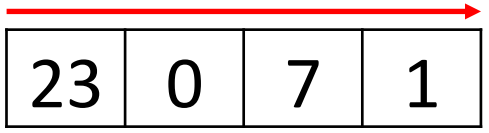
Duplicar sólo los valores de celdas en posiciones pares

23	0	7	1

46	0	14	1

Duplicar a[0] y a[2]






```
for(int i=0;i<a.length; i+=2 )
    a[i]*=2;
```

```
for(int i=0;i<a.length;i++)
{
    if(i%2!=0) continue;
    a[i]*=2;
}
```

Recorre un array en modo sólo lectura

0	1	2	3
23	0	7	1

Uso de for `int[] a = {23,0,7,1};`

Obtener la media de los valores del array y contar los pares

```
int tot = 0, con = 0;

for(int v:a)
{
    if(v%2==0) con++;
    tot+=v;
}

media = (double) tot/a.length;
```

Tipo de dato de las celdas del array

Variable que almacena el valor actual de la celda del array

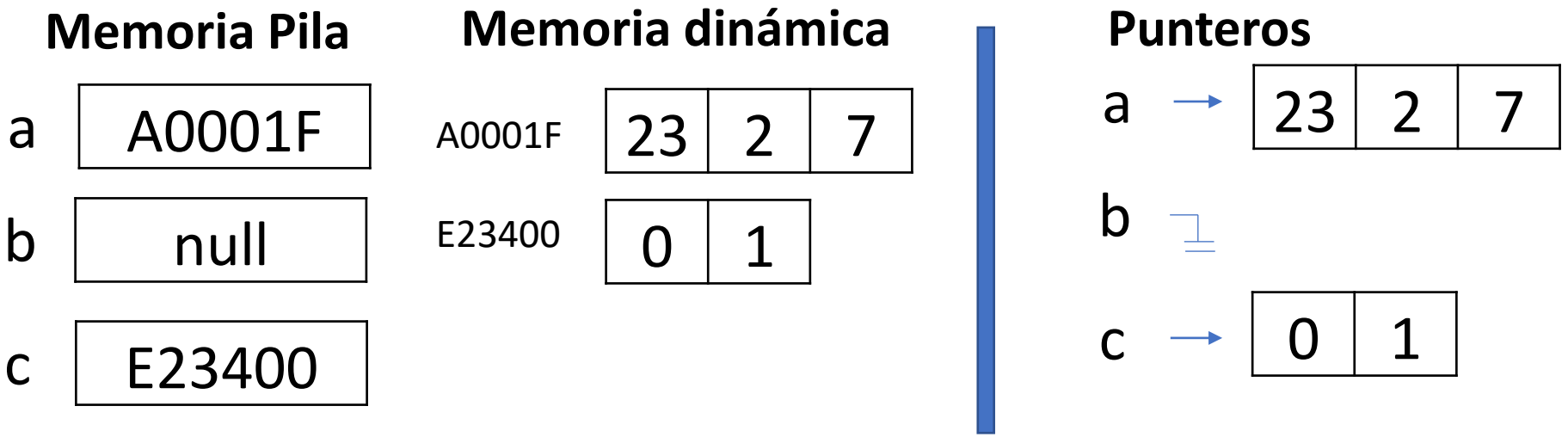
Variable del array que se quiere recorrer

No se pueden ejecutar instrucciones que modifiquen el valor de "v" como por ejemplo:
v = 2;
v += 10;

Arrays y objetos

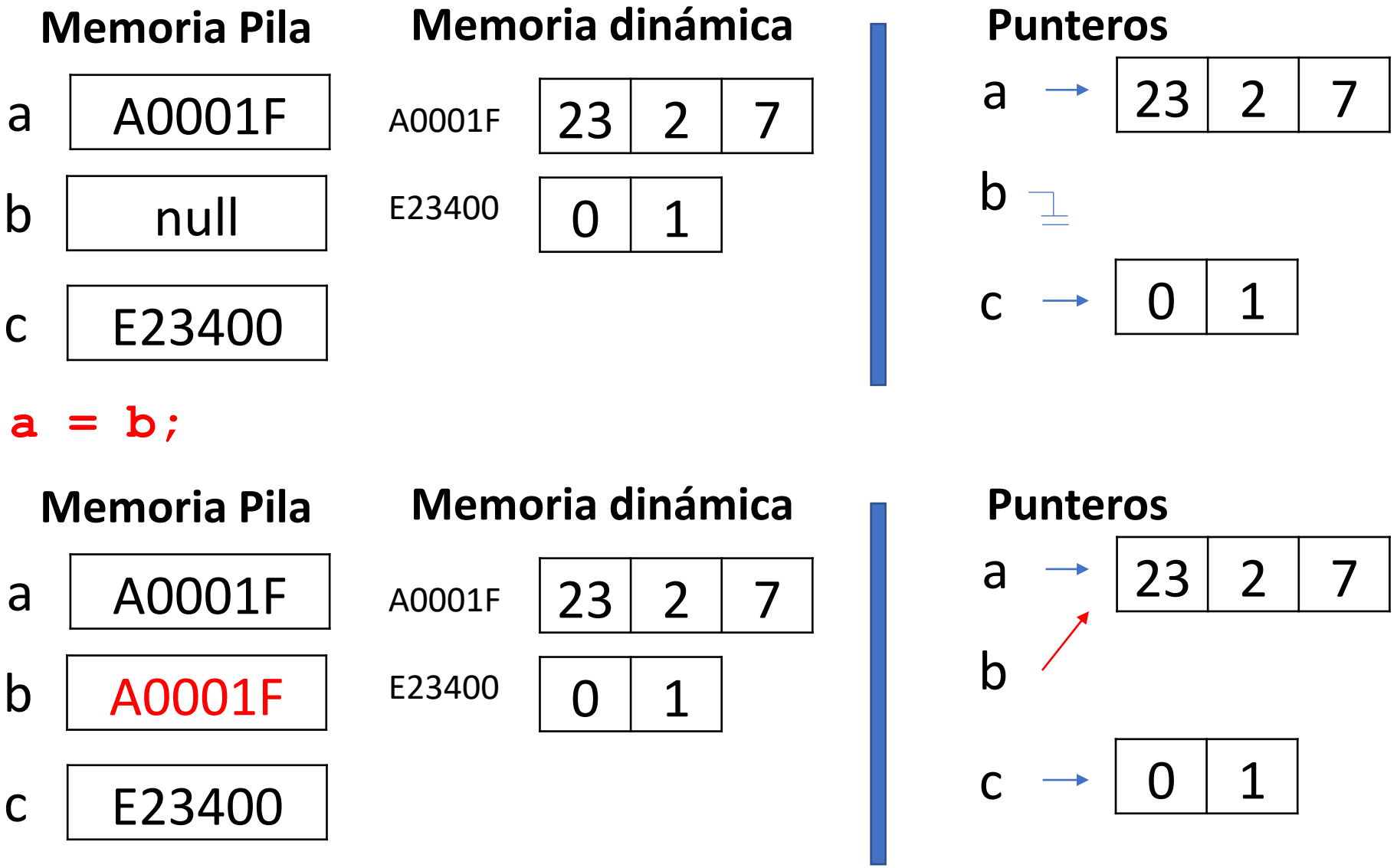
Un identificador de tipo array almacena la dirección de la primera celda del array si es distinto de null y ninguna dirección si fuera null

```
int[] a = {23,2,7}, b = null, c = {0,1};
```

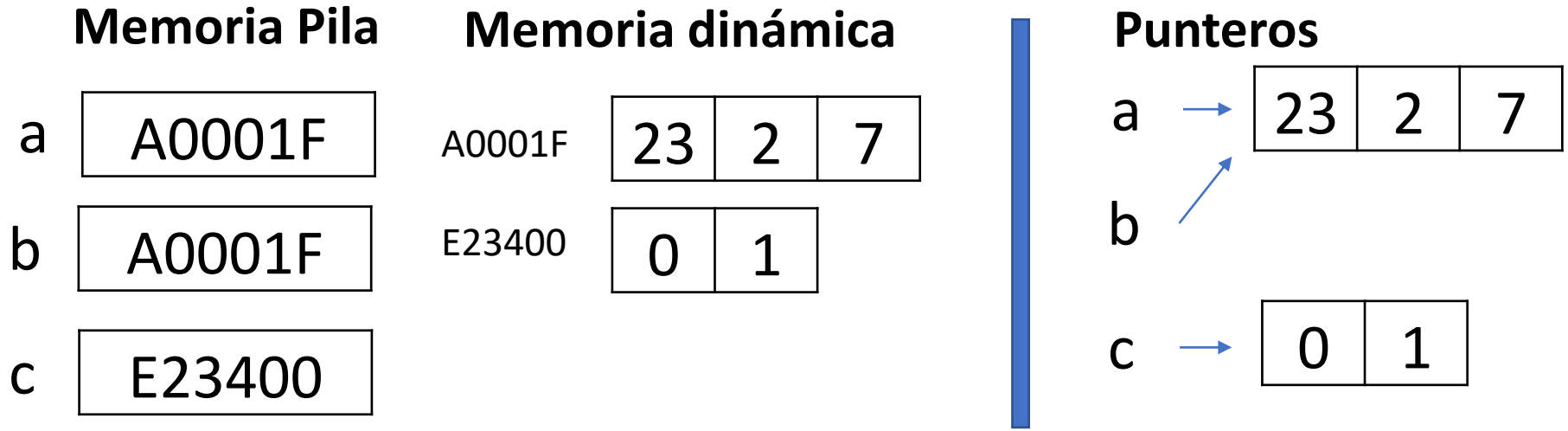


Arrays y objetos. Copiar arrays

Copiar “b” en “a” supone asignar a “a” la misma zona de memoria de “b”. Cualquier modificación del array “a” afecta también a “b”. Por ejemplo: a[2] = 3, entonces b[2] vale 3. Se dice que “a” y “b” son alias



Arrays y objetos. Reinicializar un array




```
a = new int[]{0,1};
```

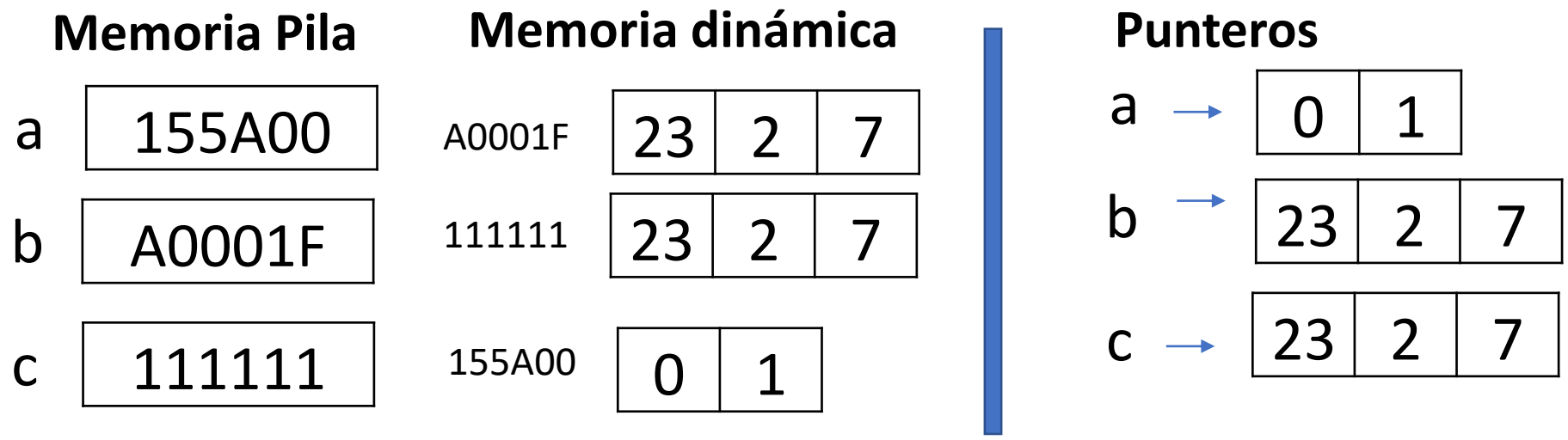
Reinicializar un array supone crea una nueva zona de memoria con los datos

Arrays y objetos. Clonar un array

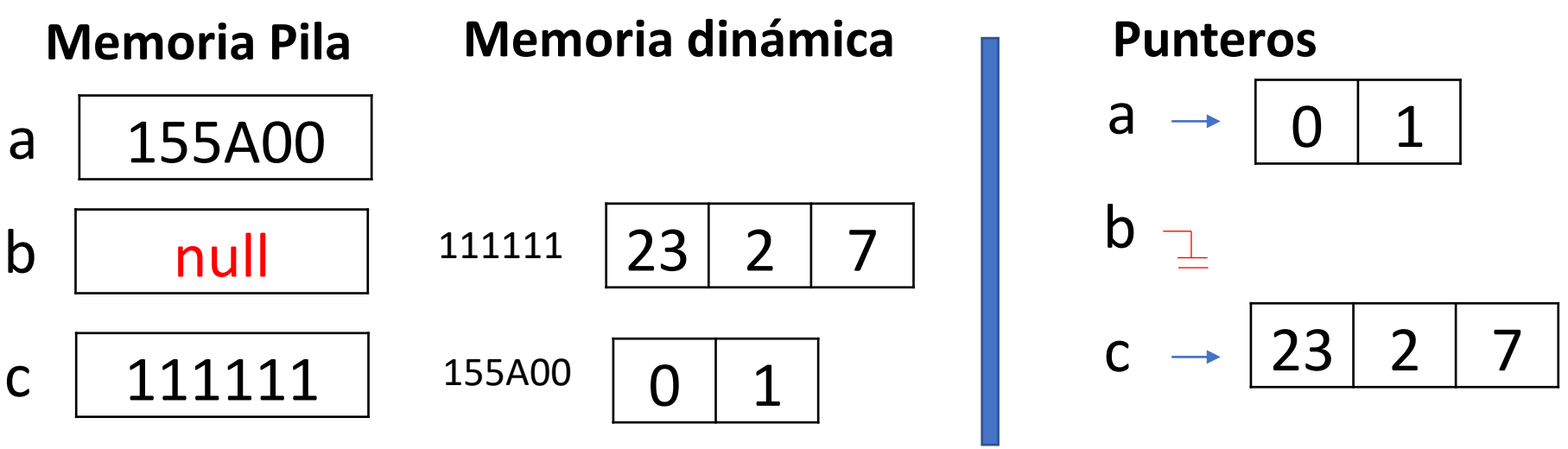

```
c = b.clone();
```

Al clonar el array “b” se genera una nueva zona de memoria para “c” con los mismos datos. Si se hace b[2] = 3, no se modifica “c” (c[2] sigue valiendo 7)

Arrays y objetos. El valor null

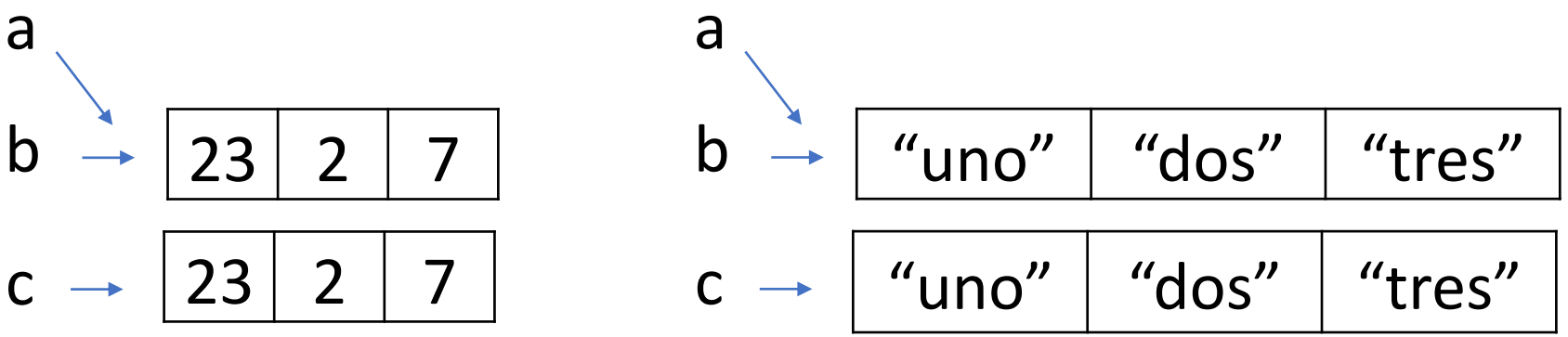


Al hacer null el array "b", la zona de memoria apuntada por "b" se libera si no hay otro array que apunte a dicha zona



El operador == y el método equals

El operador == y el método “equals” aplicado a dos arrays del mismo tipo sólo devuelven verdadero si los dos arrays son alias uno del otro.



a == b → true
a.equals(b) → true
a == c → false
a.equals(c) → false

El operador == se puede utilizar para comprobar si dos arrays son alias, es decir, apuntan a la misma zona de memoria

Métodos con arrays

Hágase un programa que rellene un array de enteros cuyo tamaño positivo se introduce por consola con valores aleatorios entre 0 y 4, ambos inclusive, lo muestre separando los datos con comas, contabilice el número de valores pares, obtenga un array con sólo los valores pares, muestre dicho array obtenido, duplique los datos del array original y lo vuelva a mostrar.

- Todos los parámetros de tipo array son por referencia (son alias de los argumentos)
- Con la notación v[] se indica que serán tratados como parámetros por valor, es decir, la modificación del parámetro no afectará al argumento
- Las funciones pueden devolver un array

Se definirán las siguientes funciones:

```
int[] aleatorio(int n)
```

```
int contarPares(int v[])
```

Se definirán los siguientes procedimientos:

```
void mostrar(int v[])
```

```
void duplicar(int[] v)
```

Métodos con arrays: aleatorio

```
static int[] aleatorio(int n)
{
    if(n<=0)
        throw new IllegalArgumentException("Tamaño invalido");

    int[] v = new int[n];

    for(int i=0;i<v.length;i++)
        v[i] = (int)(Math.random()*5);

    return v;
}
```

```
//main
int[] a = null;
int tam = 0;

tam = in.leerInt("Tamaño:
",v->v>0);

a = aleatorio(tam);
```

Métodos con arrays: mostrar

```
static void mostrar(int v[])
{
    for(int i=0;i<v.length;i++)
    {
        if(i>0) System.out.print(", ");
        System.out.print(v[i]);
    }

    System.out.println();
}
```

v = a → "v" es un alias de "a"

```
//main
int[] a = null;
int tam = 0;

tam = in.leerInt("Tamaño:
",v->v>0);

a = aleatorio(tam);

mostrar(a) ;
```

```
static void mostrar(int v[])
{
    int[] cv = v.clone();

    for(int i=0;i<cv.length;i++)
    {
        if(i>0) System.out.print(", ");
        System.out.print(cv[i]);
    }
    System.out.println();
}
```

Método alternativo para garantizar que "v" es tratado por valor. Se clona "v" y se procesa el array clonado

Métodos con arrays: contarPares

```
static int contarPares(int v[])
{
    int con = 0;

    for(int val:v)
        if(val%2==0)
            con++;

    return con;
}
```

v = a → "v" es un alias de "a"

```
//main
int[] a = null;
int tam = 0;

tam = in.leerInt("Tamaño:
",v->v>0);

a = aleatorio(tam);
mostrar(a);

int con = 0;

con = contarPares(a);
```

Método alternativo para garantizar que "v" es tratado por valor. Se clona "v" y se procesa el array clonado

```
static int contarPares(int v[])
{
    int[] cv = v.clone();

    int con = 0;

    for(int val:cv)
        if(val%2==0)
            con++;

    return con;
}
```

Métodos con arrays: duplicar

```
//main
int[] a = null;
int tam = 0;

tam = in.leerInt("Tamaño: ",v->v>0);
a = aleatorio(tam);
mostrar(a);
int con = 0;
con = contarPares(a);

duplicar(a) ;

mostrar(a);
```

La declaración e inicialización de variables y constantes se realiza cuando se precisa (no es obligatorio hacerla al inicio del programa o del método)

v = a → "v" es un alias de "a"

Salida consola

```
Tamaño: 6
4, 4, 0, 2, 1, 3
8, 8, 0, 4, 2, 6
```

```
static void duplicar(int[] v)
{
    for(int i=0;i<v.length;i++)
        v[i]*=2;
}
```

Métodos con arrays: librerias

```
public static int[] aleatorio(int n, int desde, int hasta)
{
    if(n<=0)
        throw new IllegalArgumentException("Tamaño incorrecto");

    if(desde>hasta)
    {
        int aux = desde;
        desde = hasta;
        hasta = aux;
    }

    int[] v = new int[n];
    for(int i=0;i<v.length;i++)
        v[i] = aleatorio(desde, hasta);

    return v;
}
```

Función que crea un array de enteros de tamaño “n” con números aleatorios entre “desde” y “hasta”

Invoca una sobrecarga del método “aleatorio” definido en la librería “lib”

Hay que importar fsg.lib

```
//main
int[] a = null;
int tam = 0;
tam = in.leerInt("Tamaño: ",v->v>0);
a = lib.aleatorio(tam,0,4) ;
```

Métodos con arrays: librerías (II)

Procedimiento que muestra los datos de un array de enteros separados por comas

```
public static void mostrar(int v[])
{
    for(int i=0;i<v.length;i++)
    {
        if(i>0) System.out.print(", ");
        System.out.print(v[i]);
    }

    System.out.println();
}
```

Hay que importar fsg.lib

```
//main
int[] a = null;
int tam = 0;
tam = in.leerInt("Tamaño: ",v->v>0);
a = lib.aleatorio(tam,0,4);
lib.mostrar(a) ;
```

Métodos con arrays: Seleccionar

Hágase una función que obtenga los valores pares contenidos en un array de enteros

```
static int[] obtenerPares(int v[])
{
    int[] s = new int[contarPares(v)];

    int pos = 0;
    for(int val:v)
        if(val%2==0)
        {
            s[pos] = val;
            pos++;
        }

    return s;
}
```

Se crea un array de tamaño el número de celdas de "v" con valor par (contarPares)

Se inicializa la primera posición de array "s" a rellenar con los valores pares

Si la celda actual del array "v" es par, se copia dicho valor en el array "s" en la "pos" actual y se prepara la siguiente libre

```
//main
int[] a = null, b = null;
a = lib.aleatorio(10,0,9);
lib.mostrar(a);

b = obtenerPares(a) ;

lib.mostrar(b);
```

Salida consola

```
7, 8, 9, 6, 4, 0, 4, 0, 0, 9
8, 6, 4, 0, 4, 0, 0
```

Tipos básicos y de cadena por referencia

- En java se puede simular parámetros por referencia de tipos básicos y de cadenas declarando tales tipos como arrays constantes de tamaño 1

```
int n = 3;                    →    final int n[] = {3};
String cad = "Hola";                final String cad[] = {"Hola"};
```

- Cualquier referencia a la variable “n” o “cad” se sustituiría por “n[0]” o “cad[0]”

```
n++;                    →    n[0]++;
cad = "Saludos";        →    cad[0] = "Saludos";
if(cad.equals("")) ...  →    if(cad[0].equals("")) ...
mostrar(n+1,cad);       →    mostrar(n[0]+1,cad[0]);
System.out.println(n);  →    System.out.println(n[0]);
```

Parámetros por referencia de tipos básicos

Hágase un programa que lea un entero, lo duplique y lo muestre

```
static void duplicar(int n[])
{
    if(n==null || n.length!=1)
        throw new IllegalArgumentException("Error");

    n[0]*=2;
}

static void mostrar(int n)
{
    System.out.println(n);
}
```

```
//main
final int a[] = {0};

a[0] = in.leerInt();

duplicar(a) ;

mostrar(a[0]);
```

Ambas cosas juntas indica que se trata de un parámetro de tipo básico o cadena por referencia

Se comprueba que es un array de tamaño 1

"n" es un alias de "a"

Modificar el valor del tipo básico por referencia

Paso del valor del tipo básico por referencia

Tipo básico entero por referencia

Arrays dinámicos

Los arrays dinámicos son arrays que aumentan o disminuyen de tamaño sin perder todos los datos

Hágase una función que dado un array entero y un valor entero añada dicho valor por el final del array

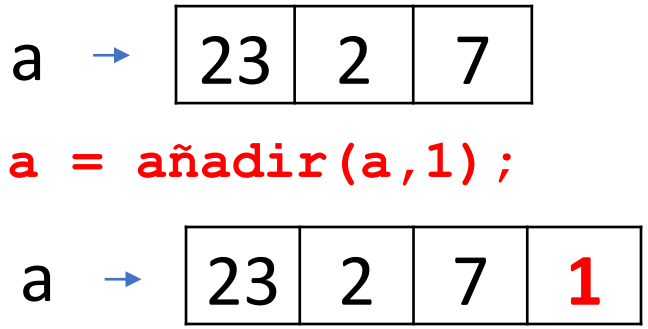
```
static int[] añadir(int v[], int n)
{
    if(v==null) v = new int[0];

    int[] s = new int[v.length+1];

    for(int i=0;i<v.length;i++)
        s[i]=v[i];

    s[s.length-1] = n;

    return s;
}
```

Si "v" es null, se reinicializa a un array de tamaño 0

Se crea un array con un elemento más que "v"

Se copia cada celda de "v" en "s"

Se copia en la última celda de "s" el valor "n"

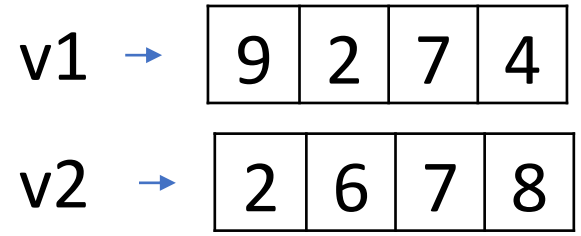


Procesamiento de múltiples arrays

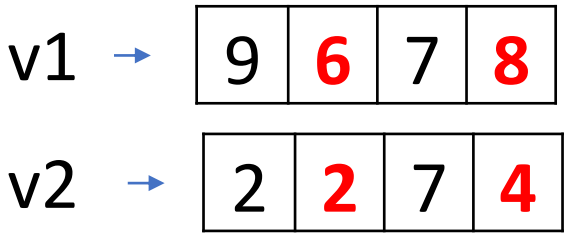
Hágase un procedimiento que intercambie las celdas con valores pares que están en la misma posición de dos arrays El procedimiento no hará nada si los dos arrays no tienen el mismo tamaño

```
static void intercambiarPares(int[] a, int[] b)
{
    if(a.length!=b.length) return;
    for(int i=0; i<a.length; i++)
        if(a[i]%2==0 && b[i]%2==0)
        {
            int aux = a[i];
            a[i] = b[i];
            b[i] = aux;
        }
}
```

Se termina la ejecución del procedimiento si ambos arrays no tienen el mismo tamaño




```
intercambiarPares(v1,v2);
```



Procesamiento de múltiples arrays

Hágase una función que sume dos arrays La función devolverá una array de tamaño 0 si los dos arrays no tienen el mismo tamaño

```
static int[] sumar(int[] a, int[] b)
{
    if(a.length!=b.length) return new int[0];

    int[] sum = new int[a.length];

    for(int i=0; i<a.length; i++)
        sum[i] = a[i]+b[i];

    return sum;
 }
```

Se devuelve una array de tamaño 0 si los dos arrays no tienen el mismo tamañaño

v1 → | 9 | 2 | 7 | 4 |

v2 → | 2 | 6 | 7 | 8 |

s = sumar(v1,v2);

s → | **11** | **8** | **14** | **12** |

Array de caracteres

Se tienen los siguientes métodos de conversión entre cadenas y arras de caracteres

Convertir la cadena “s” en un array de caracteres

`s.toCharArray()` → `char[]`

```
Si s="hola", la salida sería {'h', 'o', 'l', 'a'}
```

Convertir un array de caracteres “c” en una cadena

`String.valueOf(c)`→`String`

```
Si c={'h', 'o', 'l', 'a'}, la salida sería "hola"
```

Arrays de cadenas

Hágase un programa que lea 5 palabras. Se invertirán las cinco palabras y se mostrarán. Utilícense métodos.

```
static void leer(String[] v)
{
   for(int i=0;i<v.length;i++)
      v[i]=in.leerString();
}
static void mostrar(String v[])
{
   for(String valor:v)
      System.out.println(valor);
}
static void invertir(String[] v)
{
   for(int i=0;i<v.length;i++)
      v[i] = new StringBuffer(v[i]).reverse().toString();
}
```

```
//main
String[] s = new String[5];

leer(s);
invertir(s);
mostrar(s);
```

Salida consola

```
ESCRIBE UNA PALABRA: Hola
ESCRIBE UNA PALABRA: Retamar
ESCRIBE UNA PALABRA: FP
ESCRIBE UNA PALABRA: colegio
ESCRIBE UNA PALABRA: estudiar
aloH
ramateR
PF
oigeloc
raidutse
```

Métodos de cadenas que devuelven un array

Dividir “s” en subcadenas separadas por cadenas que cumplen una expresión regular

s.split("[0-9]+") → String[] (devuelve un array de subcadenas de “s” que están separadas por cadenas númericas)

```
Si s="uno123dos2tres", la salida sería {"uno", "dos", "tres"}
```

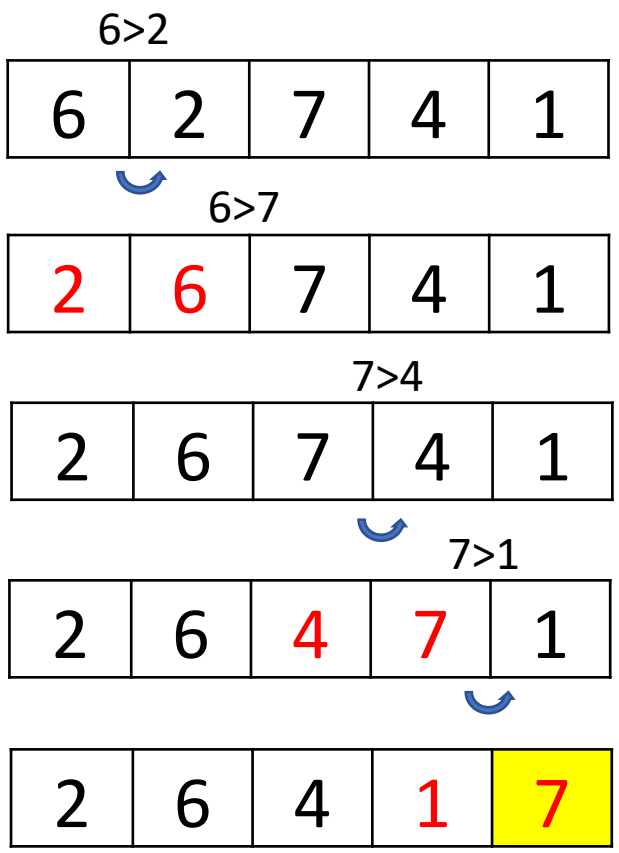
s.split("[\\|\\.@]",10) → String[] (separa la cadena “s” en subcadenas con carácter separador de ‘|’, ‘.’ O ‘@’ con un límite de 10)

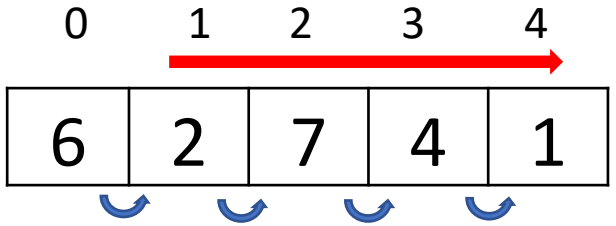
```
Si s="uno||dos@tres", la salida sería {"uno", "", "dos", "tres"}
```

Ordenar arrays básicos: Método de la burbuja

Para ordenar un array de enteros de menor a mayor se hace así:

1- Llevar el mayor valor a la última posición






```
for(int i=1;i<a.length;i++)
    if(a[i-1]>a[i])
    {
        int aux = a[i-1];
        a[i-1]=a[i];
        a[i]=aux;
    }
```

Se recorre el array de izquierda a derecha desde la posición 1. Se intercambian sucesivamente los valores si a[0]>a[1], a[1]>a[2], a[2]>a[3] y a[3]>a[4], es decir, si a[i-1] > a[i] desde i=1 hasta i=4

Criterio de ordenación

2- Repetir el proceso para llevar el mayor valor a la penúltima posición, antepenúltima, etc. (5 veces, una por posición)

```
for(int n=0;0<a.length;n++)
  for(int i=1;i<a.length;i++)
      if(a[i-1]>a[i])
      {
          int aux = a[i-1];
          a[i-1]=a[i];
          a[i]=aux;
      }
```

Ordenar arrays: Método de la burbuja mejorado

```
for(int n=0;0<a.length;n++)
  for(int i=1;i<a.length-n;i++)
     if(a[i-1]>a[i])
     {
        int aux = a[i-1];
        a[i-1]=a[i];
        a[i]=aux;
     }
```

Para ordenar ascendentemente un array básico se puede utilizar el método "sort" de la clase "Arrays" del paquete "java.útil".
Int[] a = {2,5,0,1,0};
Arrays.sort(a);
Array.sort(a, 1,3) //Ordena a[1], a[2] y a[3]

```
for(int n=0;0<a.length;n++)
{
  boolean cambio = false;
  for(int i=1;i<a.length-n;i++)
     if(a[i-1]>a[i])
     {
        int aux = a[i-1];
        a[i-1]=a[i];
        a[i]=aux;
        cambio = true;
     }
  if(!cambio) break;
}
```

Tablas

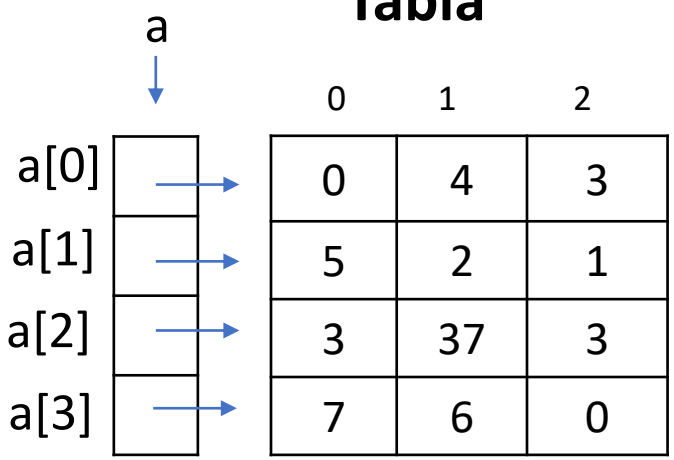
Objetivos

1) Declarar e inicializar tablas. Reinicializar. Tamaño. Acceso a las celdas de una tabla. Mostrar una tabla por consola
2) Recorrer total y parcialmente una tabla en todos los sentidos
3) Recorrer tablas dentadas
4) Copiar y clonar una tabla
5) Métodos con tablas. Librerías.

Tablas de tipos básicos y cadenas

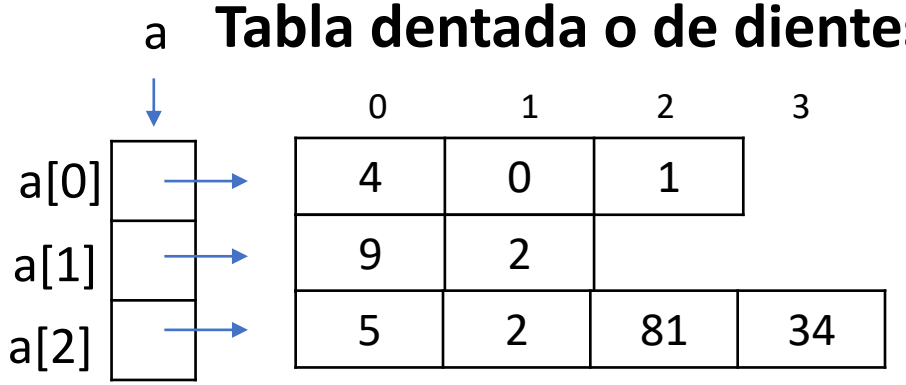
Una tabla es una representación de datos en filas (líneas horizontales) y columnas (líneas verticales)

Tabla



Es un tabla de 4 filas y 3 columnas

Tabla dentada o de dientes de sierra



Es un tabla dentada de 3 filas

Una tabla “a” es un array de arrays

Las filas son arrays referenciados como a[0], a[1], a[2], a[3]

En una tabla dentada las filas no tienen el mismo número de columnas

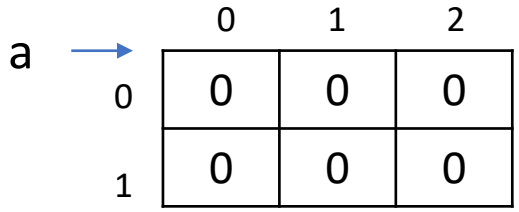
Declaración de tablas: Tamaño

```
char[][] e = null;
```

e

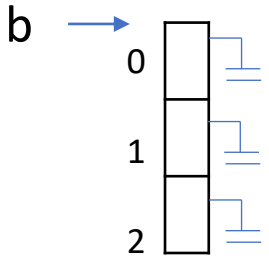
La tabla no es accesible

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



"a" tiene 2 filas y 3 columnas

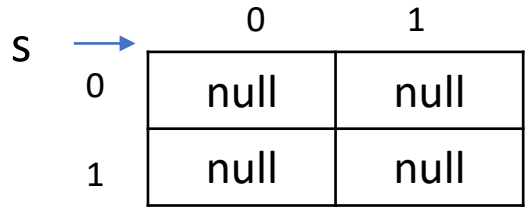
```
double[][] b = new double[3][];
```



Las filas b[0], b[1] y b[2] no son accesibles (valen null)

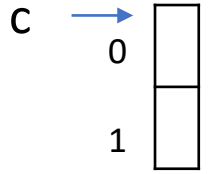
"b" tiene 3 filas sin columnas

```
String[][] s = new int[2][2];
```



"s" tiene 2 filas y 2 columnas

```
long[][] c = new long[2][0];
```



Las filas c[0] y c[1] son accesibles pero tienen 0 celdas

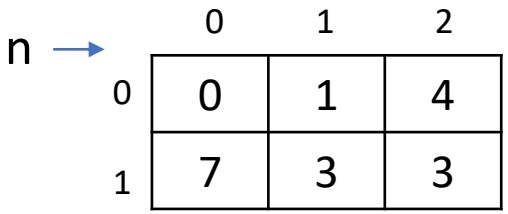
"c" tiene 2 filas y 0 columnas

También se puede utilizar la notación int a[][] para declarar una tabla

Declaración de tablas: Valores

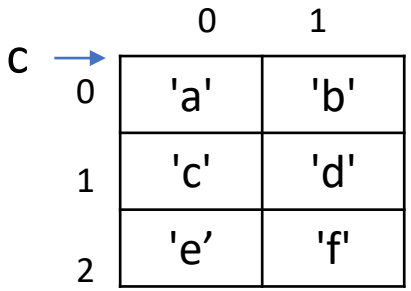
Tablas

```
int[][] n = {{0,1,4},{7,3,2}};
```



"n" tiene 2 filas y 3 columnas

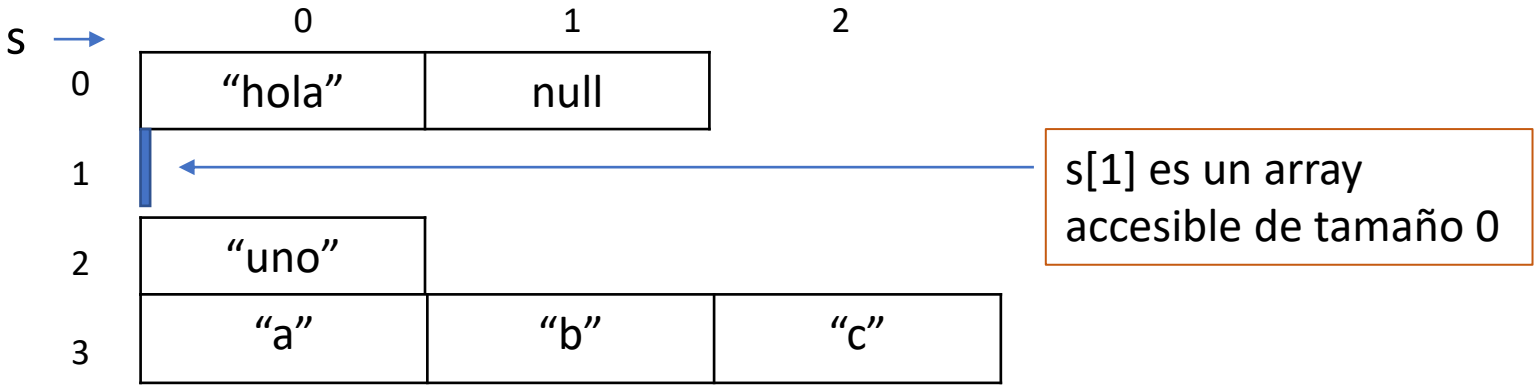
```
char[][] c = {
        {'a','b'},
        {'c','d'},
        {'e','f'}
};
```



"c" tiene 3 filas y 2 columnas

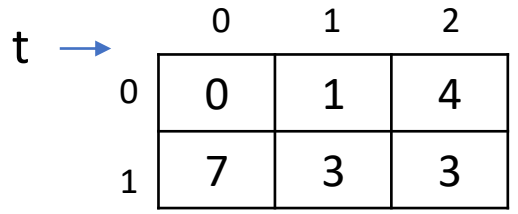
Tablas dentadas

```
String[][] s = {
{
    {"hola",null},
    {},
    {"uno"},
   {"a","b","c"}
}
```



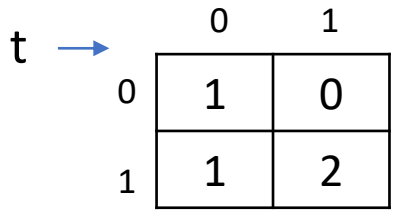
"s" tiene 4 filas

Reinicialización de una tablas

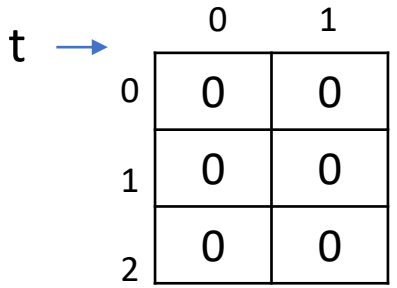


Durante la ejecución del programa es posible reinicializar una tabla o las filas de dicha tabla de alguna de las siguientes formas:

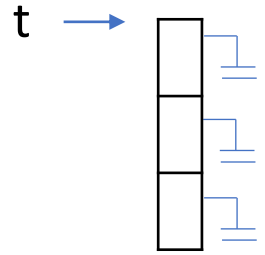
```
t = new int[][]{{1,0},{1,2}};
```



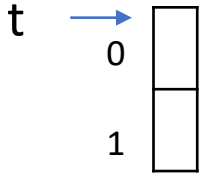

```
t = new int[3][2];
```



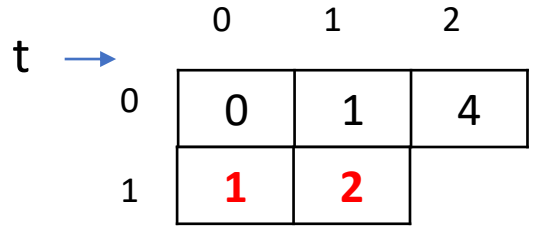

```
t = new int[3][];
```



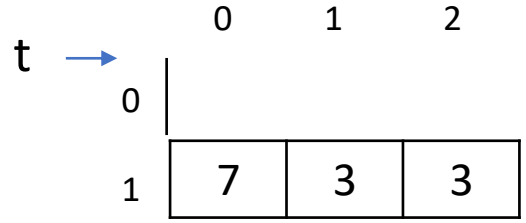

```
t = new int[2][0];
```




```
t[1] = new int[]{1,2};
```




```
t[0] = new int[0];
```

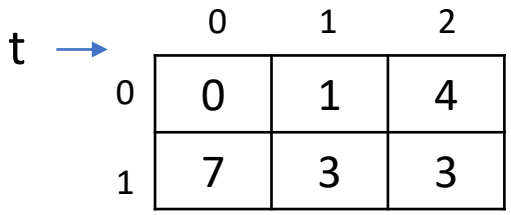


Tamaño de una tabla

A partir del nombre de una tabla se puede obtener el número de filas, y para cada fila el número de columnas de esa fila

Tablas

```
int[][] t = {{0,1,4},{7,3,2}};
```



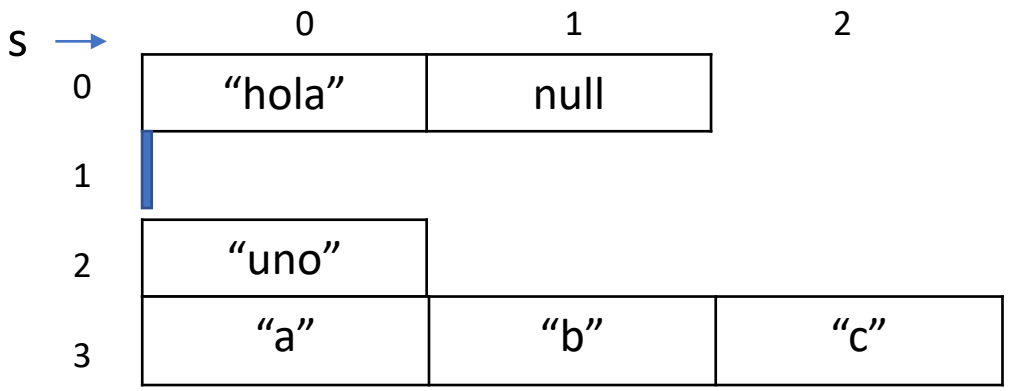
`t.length` → 2 filas

`t[0].length` → 3 columnas

En una tabla el número de columnas de cada fila coinicde

Tablas dentadas

```
String[][] s = {
{
    {"hola",null},
    {},
    {"uno"},
    {"a","b","c"}
}
```



`s.length` → 4 filas

`n[0].length` → 2 columnas

`n[1].length` → 0 columnas

Instrucciones de acceso a una tabla

Ejemplos de instrucciones sobre la tabla “t”

```
int i = 1, j = 1;
int[][] t = {{0,1,4},{7,3,2}};
```

t →

	0	1	2
0	0	1	4
1	7	3	2

```
t[0][1]++;
```

0	**2**	4
7	3	2

```
t[2*i-1][1]=t[1][j-1];
```

0	1	4
7	**7**	2

```
t[t.length-1][t[0].length-1]=10;
```

0	1	4
7	3	**10**

```
if(t[i][0]>0) t[0][j+1]+=2;
```

0	1	**6**
7	3	2

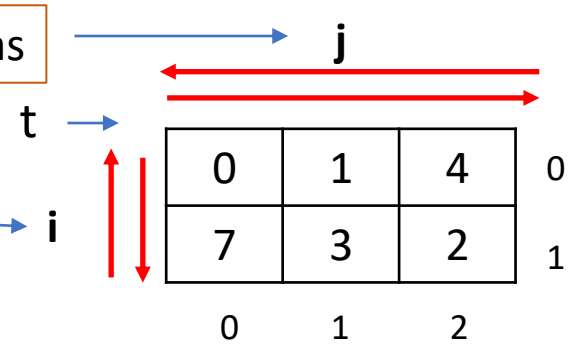
```
t[i-j][i+j]=2*t[i][2*j]+t[2-i][j-1];
```

0	1	**11**
7	3	2

Recorrer una tabla

Variable de recorrido de columnas

Variable de recorrido de filas



Uso de for

```
int[][] t = {{0,1,4},{7,3,2}};
```

De arriba a abajo

```
for(int i=0;i<t.length;i++)
```

De abajo a arriba

```
for(int i=t.length-1;i>=0;i--)
```

De izquierda a derecha

```
for(int j=0;j<t[0].length;j++)
```

De derecha a izquierda

```
for(int j=t[0].length-1;j>=0;j--)
```

Mostrar

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

i>0

Salida consola

```
0, 1, 4
7, 3, 2
```

Contar celdas con valores positivos

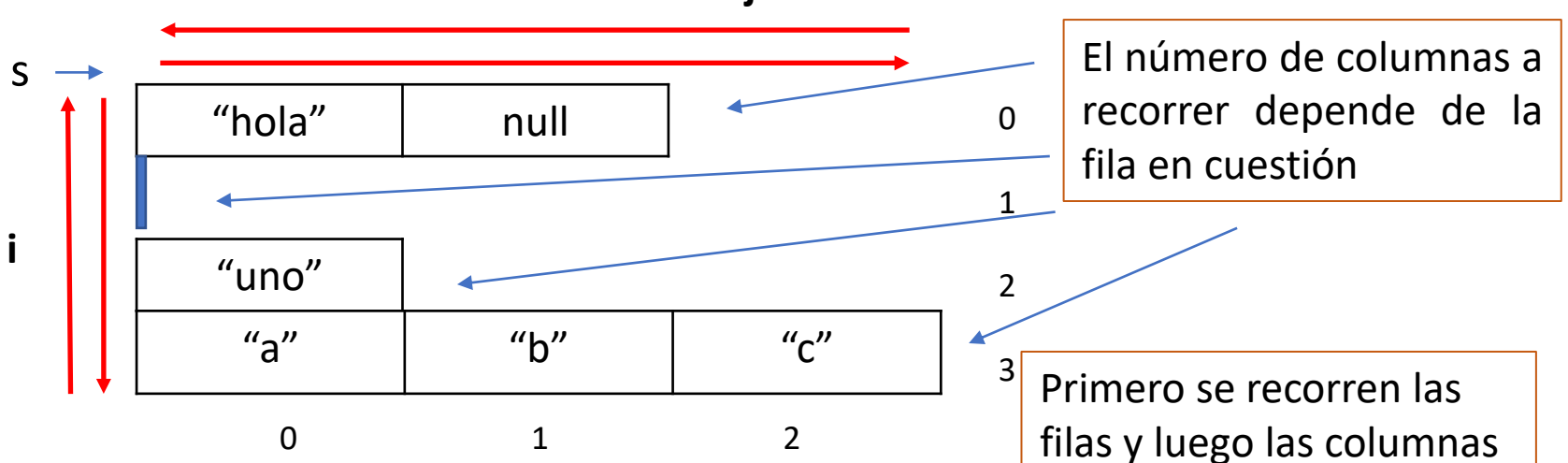
```
for(int i=0;i<t.length;i++)
    for(int j=0;j<t[0].length;j++)
        if(t[i][j]>0) con++;
```

```
0, 7
1, 3
4, 2
```

Se puede recorrer indistintamente primero las filas o las columnas. El resultado puede variar en algunos casos

Recorrer una tabla dentada

```
String[][] s = {
{
    {"hola",null},
    {},
    {"uno"},
    {"a","b","c"}
}
```



El número de columnas a recorrer depende de la fila en cuestión

Primero se recorren las filas y luego las columnas

De arriba a abajo

```
for(int i=0;i<t.length;i++)
```

De abajo a arriba

```
for(int i=t.length-1;i>=0;i--)
```

De izquierda a derecha

```
for(int j=0;j<t[i].length;j++)
```

De derecha a izquierda

```
for(int j=t[i].length-1;j>=0;j--)
```

Mostrar

```
for(int i=0;i<s.length;i++)
{
    for(int j=0;j<s[i].length;j++)
    {
        if(j>0) System.out.print(", ");
        System.out.print(s[i][j]);
    }
    System.out.println();
}
```

Salida consola

```
hola, null

uno
a, b, c
```

Cálculo de arrays asociados a una tabla

Uso de for

```
int[][] t = {{0,1,4},{7,3,2}};
```

t →

	0	1	2
0	0	1	4
1	7	3	2

Recorrer filas

```
//Fila 0
for(int j=0;j<t[0].length;j++)
   //Acceso a t[0][j]
```

Sumar los valores de cada una de las filas

```
int[] suma = null;
suma = new int[t.length];

for(int i=0;i<t.length;i++)
   for(int j=0;j<t[0].length;j++)
      sum[i] += t[i][j];
```

sum →

0	4
1	12

	0	1	2
0	0	1	4
1	7	3	2

Recorrer columnas

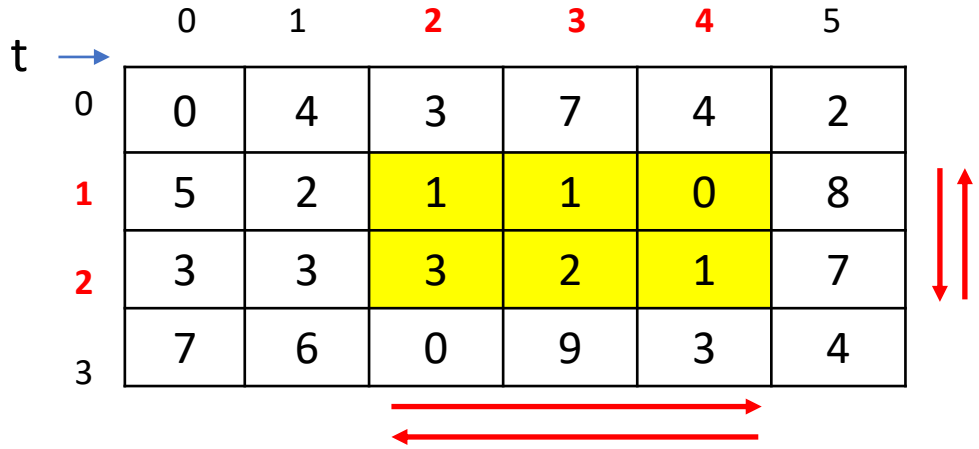

```
//Columna 2
for(int i=0;i<t.length;i++)
   //Acceso a t[i][2]
```

Sumar los valores de cada una de las columnas

```
int[] suma = null;
suma = new int[t[0].length];

for(int j=0;j<t[0].length;j++)
   for(int i=0;i<t.length;i++)
      sum[j] += t[i][j];
```

	0	1	2
	0	1	4
	7	3	2
sum →	7	4	6

Recorrer parcialmente una tabla



De arriba a abajo

```
for(int i=1;i<=2;i++)
```

De abajo a arriba

```
for(int i=2;i>=1;i--)
```

De izquierda a derecha

```
for(int j=2;j<=4;j++)
```

De derecha a izquierda

```
for(int j=4;j>=2;j--)
```

Código para sumar los valores pares de la subtabla con fondo amarillo

```
int suma = 0;
for(int i=1;i<=2;i++)
    for(int j=2;j<=4;j++)
        if(t[i][j]%2==0)
            sum+= t[i][j];
```

Recorrer una tabla en modo lectura

Uso de for

```
int[][] t = {{0,1,4},{7,3,2}};
```

t →

	0	1	2
0	0	1	4
1	7	3	2

Mostrar la tabla separando los valores de cada fila con un espacio

Tipo de dato de las celdas de la tabla (un array)

Variable que almacena la fila actual de la tabla

Variable de la tabla que se quiere recorrer

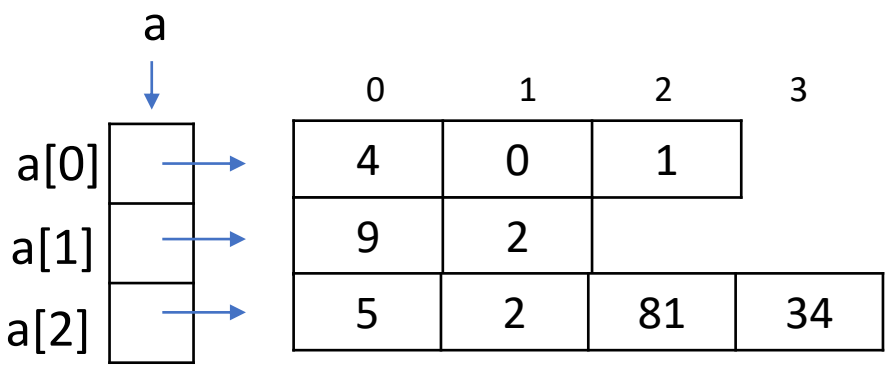
```
for(int[] f:t)
{
    for(int v:f)
        System.out.print(v+" ");
    System.out.println();
}
```

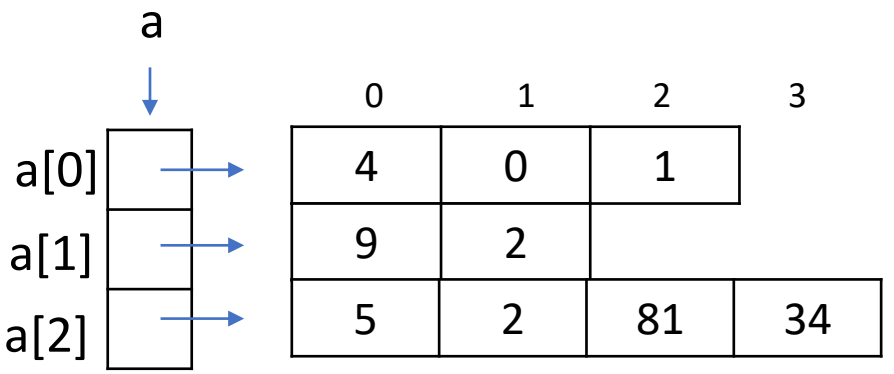
Recorrer un array (la fila en cuestión) en modo lectura

Salida consola

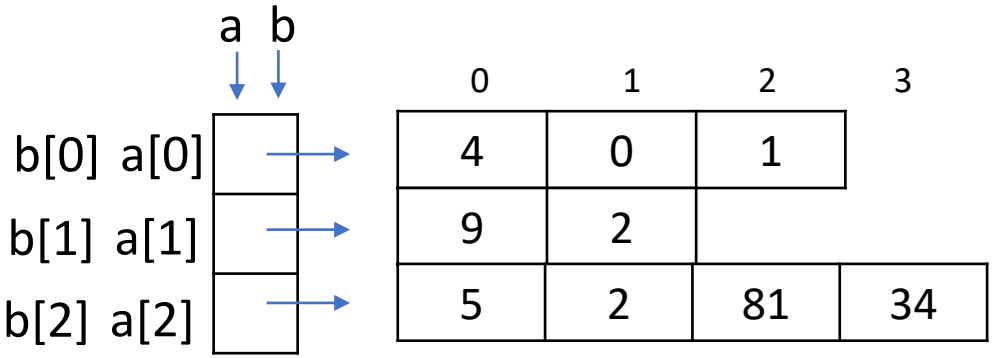
```
0 1 4
7 3 2
```

Copiar y clonar una tabla

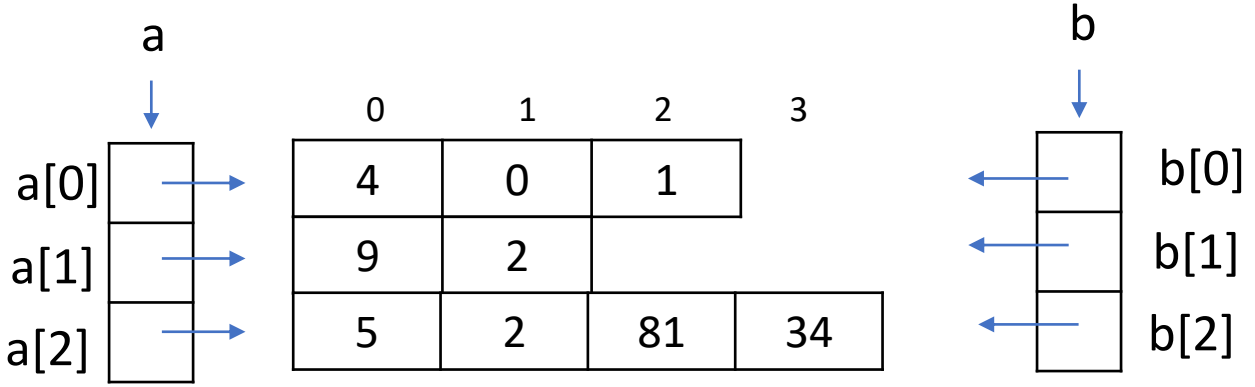





```
b = a;
```




```
b = a.clone();
```



a[0] y b[0] son alias; a[1] y b[1] son alias y a[2] y b[2] son alias.
Si hacemos a[0][1]++, entonces b[0][1] vale también 1

Métodos con tablas

Función que genera una tabla de “f” filas y “c” columnas con valores enteros aleatorios entre “d” y “h”, ambos inclusive

```
static int[][] generar(int f, int c, int d, int h)
{
   int[][] t = new int[f][c];

   for(int i=0;i<t.length;i++)
      for(int j=0;j<t[0].length;j++)
         t[i][j] = (int) (Math.random()*(h-d+1))+d;
   return t;
}
```

Función que obtiene de una tabla “t” la suma de los valores de la fila “f”

```
static int sumar(int[][] t, int f)
{
   int suma = 0;

   for(int j=0;j<t[0].length;j++)
      suma+=t[f][j];

   return suma;
}
```

Métodos con tablas (II)

Las tablas, como los arrays, se pasan como parámetros por referencia en los procedimientos

Procedimiento que incrementa en una unidad todas las celdas de una tabla “t”

Procedimiento que muestra de una tabla “t” los valores desde la fila 0 hasta la “f” y desde la columna 0 hasta la “c”

```
static void incrementar(int[][] t)
{
    for(int i=0;i<t.length;i++)
        for(int j=0;j<t[0].length;j++)
            t[i][j]++;
}
static void mostrar(int t[][], int f, int c)
{
    for(int i=0;i<=f;i++)
    {
        for(int j=0;j<=c;j++)
        {
            if(j>0) System.out.print(", ");
            System.out.print(t[i][j]);
        }
        System.out.println();
    }
}
```

Dicha notación denota que la tabla será tratada como si se pasará por valor

Librerias con tablas

Para clonar las celdas de una tabla entera se puede definir la siguiente función en la librería “lib”

```
public static int[][] clone(int[][] t)
{
    int[][] a = new int[t.length][];

    for(int i=0;i<a.length;i++)
        a[i]=t[i].clone();

    return a;
}
```

Se debería hacer una sobrecarga de dicho método para cada uno de los tipos básicos y cadena

```
//main
int[][] a = {{0,1,4},{7,3,2}};
int[][] b = null;

b = lib.clone(a);
a[0][0]++;
System.out.println(a[0][0]);
System.out.println(b[0][0]);
```

Salida consola

```
1
0
```

Clases envoltorio

Objetivos

1) Conocer la tipos envoltorio asociado a los tipos básicos. Rango y tamaños de los tipos envoltorios
2) Declaración e inicialización de tipos envoltorios. Entrada y salida
3) Conversión de tipos básicos y su correspondiente tipo envoltorio
4) Expresiones de tipos envoltorio. Conversión entre tipos envoltorio. Conversión de tipos envoltorio a tipos básicos
5) Conversión entre tipos envoltorio y cadenas
6) Conversiones a binario, octal y hexadecimal. Máximo y mínimo. Infinite y NaN
7) Ordenar arrays de tipo envoltorio
8) Métodos con arrays

Clases envoltorio

En Java se tiene asociado a cada tipo de dato básico una clase envoltorio definida en el paquete “java.lang”

TIPO BASICO	TIPO ENVOLTORIO
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

- Cada tipo envoltorio tiene el mismo rango de valores que su correspondiente tipo básico y el valor **null**
- Como en el caso de los tipos básicos sólo se pueden pasar tipos envoltorios por valor a un método

Rango de valores de tipos envoltorio

Los valores máximos y mínimos de cada uno de los tipos envoltorio se almacenan en las siguientes constantes

MINIMO	MAXIMO
Byte.***MIN_VALUE***	Byte.***MAX_VALUE***
Short.***MIN_VALUE***	Short.***MAX_VALUE***
Integer.***MIN_VALUE***	Integer.***MAX_VALUE***
Long.***MIN_VALUE***	Long.***MAX_VALUE***
Float.***MIN_VALUE*** Float.***MIN_NORMAL***	Float.***MAX_VALUE***
Double.***MIN_VALUE*** Double.***MIN_NORMAL***	Double.***MAX_VALUE***
(int) Character.***MIN_VALUE***	(int) Character.***MAX_VALUE***

Se puede asignar a una variable de tipo básico su correspondiente valor mínimo o máximo

```
//main
int a = Integer.MIN_VALUE;
double b = Double.MAX_VALUE;

System.out.println(a);
System.out.println(b);
```

Salida consola

```
-2147483648
1.79769313486231570E308
```

Tamaño de los tipos envoltorios

El número de bytes y bits que ocupa cada uno de los tipos de datos básicos se almacenan en las siguientes constantes

BYTES	BITS
Byte.***BYTES***	Byte.***SIZE***
Short.***BYTES***	Short.***SIZE***
Integer.***BYTES***	Integer.***SIZE***
Long.***BYTES***	Long.***SIZE***
Float.***BYTES***	Float.***SIZE***
Double.***BYTES***	Double.***SIZE***
Character.***BYTES***	Character.***SIZE***

Las constantes que almacenan el número de bytes y bits son de tipo byte

```
//main
byte a = Integer.BYTES;
byte b = Double.SIZE;

System.out.println(a);
System.out.println(b);
```

Salida consola

4
64

Declaración e inicialización de tipos envoltorio

La declaración e inicialización de variables y constantes de tipos envoltorio es exactamente igual que en los tipos básicos, salvo que se le puede asignar como valor de un tipo envoltorio el null

```
//main
Byte uno = 1;
Short n = 101;
Integer a = 23;
Long numero = 64546465400L;
Float velocidad = 23.78F;
Double peso = null;
Character letra = 'A';
Boolean conducir = false;
```

```
//main
Integer a = 23, b = 34;
Double peso = 90, altura = 1.78;

Float[] datos = {45, null , 56.02F};
Double v = new Double[34];

Integer[][] n = new Integer[3][6];
```

El valor asignado a un tipo envoltorio tiene que ser un literal del mismo tipo: `Double valor = 34;`

```
error: incompatible types: int
cannot be converted to Double
```

Entrada y salida de tipos envoltorio

- La lectura y validación de datos de tipo envoltorio se hace igual que con los tipos básicos
- Para la salida de datos de tipo envoltorio se puede utilizar los métodos “print”, “println” y “printf”

```
//main
Integer a = 23;
Double b = 90.0;

a = in.leerInt("ENTERO: ",v->v>0);
b = in.leerDouble("REAL: ");
System.out.println("valor: "+a);
System.out.printf("entero = %d real = %1.1f\n", a,b);
```

Salida consola

```
ENTERO: 56
REAL: 5,67
valor: 56
entero = 56 real = 5,7
```

Conversion de tipos básicos y envoltorio

El operador “=” (de asignación) permite asignar a una a variable de tipo básico un valor de una variable de tipo su envoltorio si es distinto de null (si no, se lanza una excepción), y viceversa

```
//main
int a = 23;
int aux = 0;
Integer b = 3;

a = in.leerInt();
b = in.leerInt();
aux = a;
a = b;
b = aux;
System.out.println(a);
System.out.println(b);
```

Salida consola

```
ESCRIBE UN ENTERO: 45
ESCRIBE UN ENTERO: 230
230
45
```

```
//main
int a = 23;
Integer b = null;

a = b;
System.out.println(a);
System.out.println(b);
```

Salida consola

```
Exception in thread "main"
java.lang.NullPointerException: Cannot
invoke "java.lang.Integer.intValue()"
because "b" is null
```

Expresiones de tipos de envoltorio

Se pueden utilizar los siguiente operadores entre los tipos Byte, Short, Integer, Long, Float, Double y Character:

- Operadores aritméticos: ++, --, +, -, *, /, %.
- Operadores de bits: <<, >>, >>>, &, |, ^
- Operadores de comparación: >, <, >=, <=, ==, !=
- Operadores de acumulación: =, +=, -=, *=, /=, %=, &=, !=, <<=, >>=, >>>=
- NO SE PUEDE UTILIZAR EL OPERADOR () DE CONVERSION

Los operadores de bits y sus correspondientes operadores de acumulación no se pueden utilizar con valores de tipos Float y Double

Conversión entre tipos envoltorios

La conversión entre tipos Byte, Short, Integer, Long, Float y Double se realiza utilizando los métodos “byteValue”, “shortValue”, “intValue”, “longValue”, “floatValue” y “doubleValue”.

```
//main
Byte a = 56;
Integer b = 3478;
Double c = 90.7;

b = in.leerInt();
c = in.leerDouble();
a = b.byteValue();
b = c.intValue();
System.out.println(a);
System.out.println(b);
```

Salida consola

```
ESCRIBE UN ENTERO: 122
ESCRIBE UN REAL: 3,78
122
3
```

Conversión de tipos envoltorio a tipos básicos

- Las variables de tipos envoltorio se convierten a tipos básicos utilizando los métodos “xxxValue”.
- Las expresiones de tipo envoltorio se convierten a tipos básicos utilizando el operador de conversión ()

```
//main
Double a = 0D;
Double b = 0D;
int c = 0;

a = in.leerDouble();
b = in.leerDouble();

c = a.intValue();

System.out.println(c);

c = (int)(a*b);

System.out.println(c);
```

Salida consola

```
ESCRIBE UN REAL: 5,78
ESCRIBE UN REAL: 4,32
5
24
```

Da error la instrucción:
c = (int) a;

Da error la instrucción:
c = (a*b).intValue();

Conversión de tipos envoltorio a cadenas

- El método “toString” de cada tipo de envoltorio
- El método “String.valueOf” de la cadenas
- El operador “+” tomando como primero valor el de una cadena

```
//main
Integer a = 11; Double b = 34.7; Character c = 'A';
String sa = "", sb = "", sc = "";
sa = a.toString();
sb = String.valueOf(b);
sc = ""+c;
System.out.println(sa);
System.out.println(sb);
System.out.println(sc);
System.out.println("Entero: "+a+" Real: "+b+" Caracter: "+c);
```

Salida consola

```
11
34.7
A
Entero: 11 Real: 34.7 Caracter: A
```

Conversión de cadenas a tipos envoltorio

En cada tipo envoltorio (salvo en el de Character) se tiene el método "valueOf" que sirve para convertir una cadena en un valor de dicho tipo. Si la cadena no se pudiera convertir, se lanzaría un error

```
//main
String s = "";
Double a = 0.0;

s = in.leerString("Real: ");
a = Double.valueOf(s);
System.out.println(a);
```

Salida consola

```
Real: 45.78
45.78
```

Salida consola

```
Real: 45,78
Exception in thread "main" java.lang.NumberFormatException: For input string: "45,78"
   at java.base/jdk.internal.math.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:20
   at java.base/jdk.internal.math.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
   at java.base/java.lang.Double.parseDouble(Double.java:549)
   at java.base/java.lang.Double.valueOf(Double.java:512)
```

En el caso del tipo Character se utiliza el método charAt(0) de las cadenas para extraer el primer carácter de una cadena:

String s = "Hola";
Character c = null;
c = s.charAt(0);

Conversión de cadenas a tipos de envoltorio (II)

También se puede convertir una cadena a un tipo de envoltorio utilizando los métodos “parseByte”, “parseShort”, “parseInt”, “parseLong”, “parseFloat”, “parseDouble” y “parseBoolean” de cada tipo de envoltorio. En los 4 primeros se puede indicar la base de la cadena numérica a convertir.

```
//main
Byte y = 0;
Short s = 0;
Integer i = 0;
Long l = 0L;
Double d = 0.0;
Boolean b = false;
```

Si la cadena no se pudiera convertir o el resultado de la conversión fuera un valor que está fuera del rango del tipo envoltorio se lanza un error del tipo `java.lang.NumberFormatException`

```
y = Byte.parseByte("1100",2);
s = Short.parseShort("AF",16);
i = Integer.parseInt("4502");
l = Long.parseLong("5023",6);
d = Double.parseDouble("345.49");
b = Boolean.parseBoolean("TrUe");
```

← Cadena en base 2
← Cadena en base 16
← Cadena en base 10
← Cadena en base 6

```
System.out.println(y);
System.out.println(s);
System.out.println(i);
System.out.println(l);
System.out.println(d);
System.out.println(b);
```

Salida consola →

```
12
175
4502
1095
345.49
true
```

Conversión de Integer y Long a base 2, 8 y 16

En los tipos de envoltorio Integer y Long se tienen los métodos “toBinaryString”, “toOctalString” y “toLongString” que convierte un entero en una cadena numérica en base 2, 8 y 16, respectivamente

```
//main
int a = 0;   //o Integer
Long b = 0L; //o long
String sa = "", sb = "";

a = in.leerInt();
b = in.leerLong();
sa = Integer.toBinaryString(a);
sb = Long.toHexString(b);
System.out.println(sa);
System.out.println(sb);
```

Salida consola

```
ESCRIBE UN ENTERO: 23
ESCRIBE UN ENTERO LARGO: 76524220
10111
48faabc
```

Máximo y mínimo de tipos envoltorio

En los tipos “Integer”, “Long”, “Float” y “Double” se tienen los métodos “max” y “min” para obtener el máximo y mínimos de dos enteros o reales, respectivamente.

```
//main
int a = 0;  //o Integer
Integer b = 0; //o int
Integer max = 0; //o int
int min = 0; //o Integer

a = in.leerInt();
b = in.leerInt();
max = Integer.max(a,b);
min = Integer.min(a,b);
System.out.println(max);
System.out.println(min);
```

Salida consola

```
ESCRIBE UN ENTERO: 78
ESCRIBE UN ENTERO: -234
78
-234
```

Infinito y no es un número

En los tipos “Float” y “Double” se tienen los métodos “isInfinite” y “isNaN” para comprobar si el número real obtenido es infinito o no es un número, respectivamente.

```
//main
double a = 0;  //o Double
Double b = 0D; //o double
Double v = 0D; //o double

a = in.leerDouble();
b = in.leerDouble();
v = a/b;
System.out.println(v.isNaN());
System.out.println(Double.isNaN(v));
System.out.println(v.isInfinite());
System.out.println(Double.isInfinite(v));
```

Salida consola

```
ESCRIBE UN REAL: 9
ESCRIBE UN REAL: 0
false
false
true
true
```

9/0 No es un real desconocido

9/0 Es infinito

Salida consola

```
ESCRIBE UN REAL: 0
ESCRIBE UN REAL: 0
true
true
false
false
```

0/0 No es un número

0/0 No es infinito

La interfaz Comparator

La interfaz “Comparator”, que está en el paquete “java.útil”, se utiliza para definir un criterio de comparación para un tipo envoltorio u objeto

Declaración e inicialización de un criterio de comparación entre enteros

```
//main
final Comparator<Integer> MENOR_MAYOR = (a,b) ->
{
    if(a<b) return -1;
    if(a>b) return 1;
    return 0;
};
int n1 = 0, n2 = 0;
n1 = in.leerInt();
n2 = in.leerInt();

System.out.println( MENOR_MAYOR.compare(n1,n2) );
```

Enteros a comparar. Parámetros de entrada del método “compare”. No pueden haber variables con los nombres “a” y “b”

```
ESCRIBE UN ENTERO: 5
ESCRIBE UN ENTERO: 8
-1
```

El método “compare” si devuelve -1, entonces el primer argumento es menor que el segundo; si devuelve 1, el primero es mayor que el segundo y si devuelve 0, son iguales

Otros métodos de comparación entre enteros

```
final Comparator<Integer> PARES_IMPARES = (a,b) ->
{
    if(a%2==0 && b%2==1) return -1;
    if(a%2==1 && b%2==0) return 1;
    return 0;
};
```

Primero los pares y luego los impares

```
final Comparator<Integer> POSITIVOS_NEGATIVOS_ORDENADOS = (a,b) ->
{
    if(a>0 && b<0) return -1;
    if(a<0 && b>0) return 1;
    if(a>0 && b>0 || a<0 && b<0)
    {
        if(a<b) return -1;
        if(a>b) return 1;
    }
    return 0;
};
```

Primero los positivos ordenados ascendentemente y luego los negativos ordenados ascendentemente

Ordenar Arrays de tipo envoltorio

Para ordenar Arrays de tipo envoltorio se utiliza el método “sort” de la clase “Arrays”, que admite el array a ordenar y adicionalmente un criterio de comparación

```
static void mostrar(Integer[] a)
{
    for(int i=0;i<a.length;i++)
    {
        if(i>0) System.out.print(", ");
        System.out.print(a[i]);
    }
    System.out.println();
}
```

Falta declarar e inicializar el criterio de comparación POSITIVOS_NEGATIVOS_ORDENADOS

```
//main
Integer[] v = {28,-3,2,18,-9,6,8};
mostrar(v);
Arrays.sort(v,POSITIVOS_NEGATIVOS_ORDENADOS);
mostrar(v);
Arrays.sort(v);
mostrar(v);
```

Primero los positivos y luego los negativos

Se ordenan en el orden natural de tipo del Array

Salida consola

```
28, -3, 2, 18, -9, 6, 8
2, 6, 8, 18, 28, -9, -3
-9, -3, 2, 6, 8, 18, 28
```

Métodos con tipos envoltorio

Todos los parámetros de tipo envoltorio son por valor

Comprobar si un entero es par

```
static boolean esPar(Integer a)
{
   return a%2==0;
}
```

Duplicar un entero

```
static Integer duplicar(Integer a)
{
   return 2*a;
}
```

Sumar dos reales

```
static Float sumar(Float a, Float b)
{
   return a+b;
}
```

Truncar un número real

```
static Double truncar(Double a, int pos)
{
   Double f = Math.pow(10,pos);
   return (int)(a*f)/f;
}
```

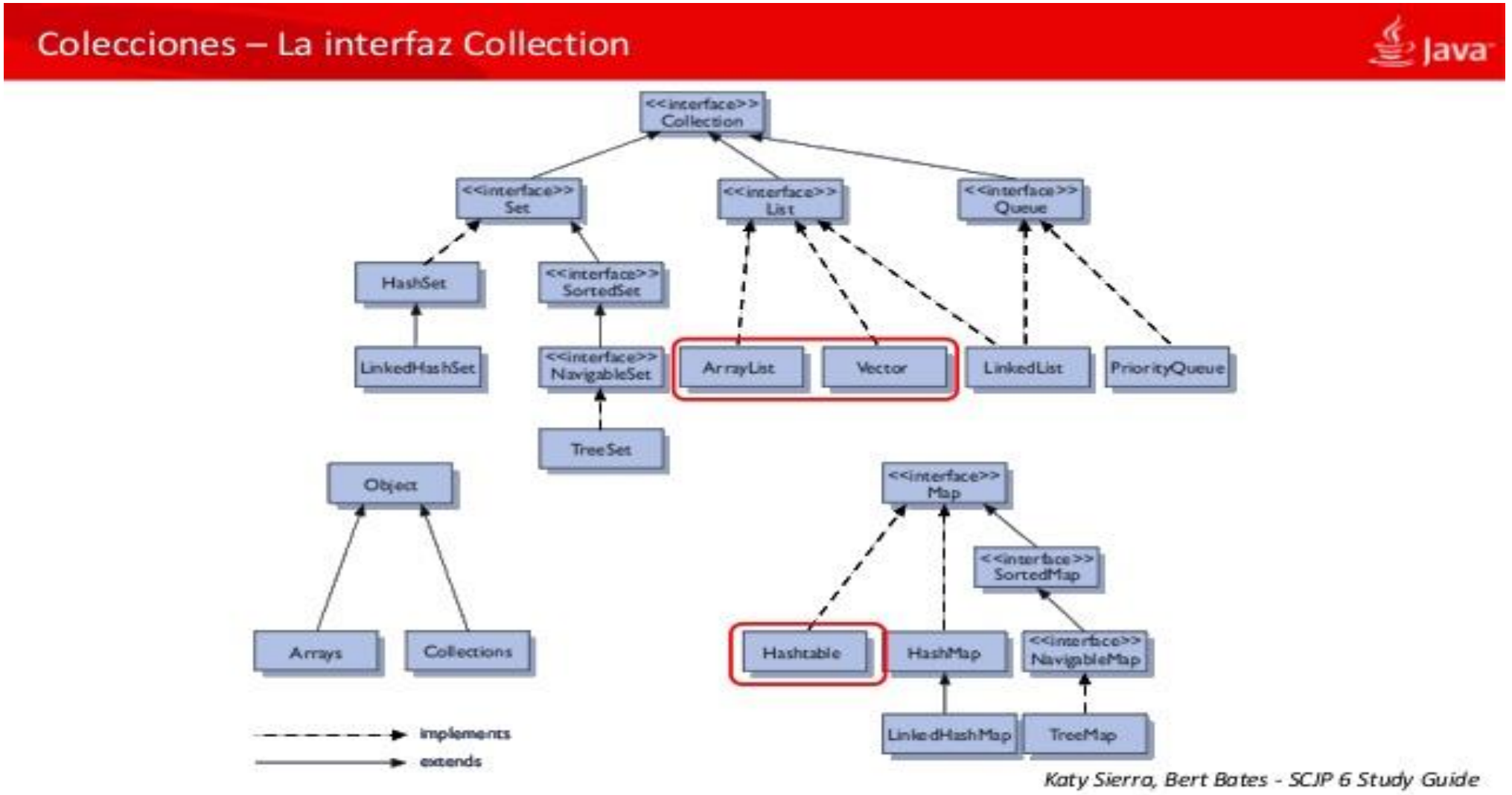
Colecciones

Objetivos

1) Saber qué es una colección
2) Manejo de ArrayList
3) Manejo de HashSet
4) Manejo de Stack
5) Manejo de Hashtable

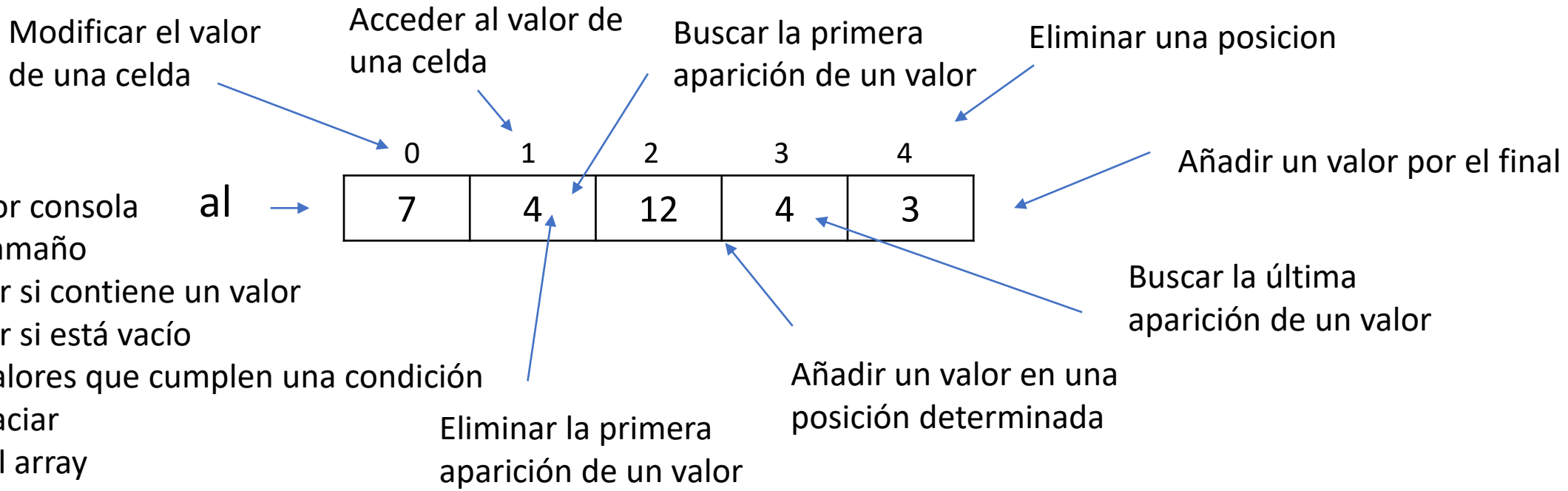
Colecciones

Se muestra una imagen con todas las colecciones que hay definidas en java



ArrayList

En Java se tiene una clase genérica “ArrayList” definida en el paquete “java.util”, que permite almacenar objetos de cualquier tipo en forma de un array dinámico, es decir, arrays que aumentan y disminuyen de forma automática al añadir y quitar elementos



- Mostrar por consola
- Obtener tamaño
- Comprobar si contiene un valor
- Comprobar si está vacío
- Eliminar valores que cumplen una condición
- Clonar y vaciar
- Recorrer el array

Crear, añadir y mostrar ArrayList

Crear un arrayList de enteros vacío

```
ArrayList<Integer> al = new ArrayList<Integer>();
```

Añadir un entero “n” a “v” por el final

```
al.add(n);
```

Mostrar por consola “al”

```
System.out.println(al);
```

Añadir un entero “n” a “al” en una posición “a”

```
al.add(a,n);
```

Hágase un programa que rellene un arrayList de enteros con valores aleatorios entre 0 y 9, y lo muestre

```
//main
ArrayList<Integer> al = new ArrayList<Integer>();

for(int i=0;i<5;i++)
{
    int n = (int)(Math.random()*10);
    al.add(n);
}
System.out.println(al);
```

Salida consola

```
[7, 2, 4, 3, 2]
```

Acceso a celdas y tamaño de un ArrayList

Obtener el valor de "al" en la posición "i"

`al.get(i)` → `Tipo del contenido`

Asignar "n" en la posición "i" de "al"

`al.set(i,n);`

Se lanza una excepción si la posición "i" no es válida

Comprobar si "al" contiene el valor "n"

`al.contains(n)` → `boolean`

Vedadero si "v" contiene el valor "n"; falso, en otro caso

Tamaño de "v"

`al.size(n)` → `int`

Ver si "al" está vacio

`al.isEmpty()` → `boolean`

Programa que muestra el tamaño de "al" y el valor de la primera celda; si "al" contiene el valor 2, incrementa la última celda en una unidad y se muestra de nuevo "al"

al → `[7, 2, 4, 3, 2]`

```
//main
int n = 0;
n = al.get(0);
System.out.println(n);
if(al.contains(2))
{
    n = al.get(al.size()-1);
    n++;
    al.set(al.size()-1,n);
}
System.out.println(al);
```

Salida consola

```
7
[7, 2, 4, 3, 3]
```

Buscar, clonar y vaciar en un ArrayList

Buscar la posición de la primera aparición de un valor "n" en "al"

```
al.indexOf(n)) → int
```

devuelve -1 si no existe el valor buscado

Buscar la posición de la última aparición de un valor "n" en "al"

```
al.lastiIndexOf(n)) → int
```

devuelve -1 si no existe el valor buscado

Vaciar `al.clear();`

Clonar

Se convierte al tipo del arrayList

```
al1 = (ArrayList) al.clone();
```

Programa que busca en "al" la primera aparición del valor "6" y la última aparición del valor "2". Se clonará "al". Se vaciará "al" y se mostrarán los dos arraysList.

al → [7, 2, 5, 3, 2]

```
//main
ArrayList al1 = null;
int pos = 0;
pos = al.indexOf(6);
System.out.println(pos);
pos = al.lastIndexOf(2);
System.out.println(pos);
al1 = (ArrayList) al.clone();
al.clear();
System.out.println(al);
System.out.println(al1);
```

Salida consola

```
-1
4
[]
[7, 2, 5, 3, 2]
```

Eliminar celdas de un ArrayList

Eliminar en “al” la celda que ocupa la posición “i” (int)

```
a = al.remove(i);
```

Eliminar la primera aparición del valor “n” en “al”

```
al.remove(n);
```

Eliminar en “al” valores que cumplen una determinada condición (valores pares)

```
al.removeIf(v->v%2==0);
```

Programa que elimina en “al” desde la cuarta hasta la sexta celda, la primera aparición del valor 5 y todos las celdas con valores pares. Se muestra “al”

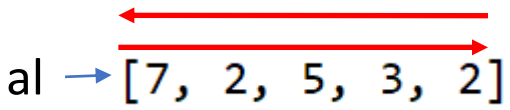
```
//main          al → [7, 2, 5, 3, 2, 0, 1, 5, 4]
Integer val = 5;
al.remove(6);
al.remove(5);
al.remove(4);
System.out.println(al);
al.remove(val);
System.out.println(al);
al.removeIf(v->v%2==0);
System.out.println(al);
```

Se elimina las posiciones de derecha a izquierda

Salida consola

```
[7, 2, 5, 3, 5, 4]
[7, 2, 3, 5, 4]
[7, 3, 5]
```

Recorrer un ArrayList



Recorrer “al” de izquierda a derecha

```
for(int i=0; i<v.size(); i++)
```

Recorrer “al” de derecha a izquierda

```
for(int i=v.size()-1; i>=0; i--)
```

Recorrer “al” en modo lectura

```
for(int v:al)  for(Integer v:al)
```

Recorrer “al” iterando

```
for(Iterator<Integer> r = al.iterator(); r.hasNext();){
      n=r.next();
}
```

La clase “Iterator” está en el paquete “java.útil”

Programa que duplica los valores de “al” y lo muestra separando sus valores con coma

al → [7, 2, 5, 3, 2]

```
//main
String s = "";
for(int i=al.size()-1;i>=0;i--)
    al.set(i,al.get(i)*2);

for(int i=0;i<al.size();i++)
{
    if(i>0) s+=", ";
    s+=al.get(i);
}
System.out.println(s);
```

Salida consola

14, 4, 10, 6, 4

Ordenar ArraysList

Para ordenar ArraysList se utiliza el método “sort” de la clase “Collections” (definida en el paquete “java.útil”), que admite el ArrayList a ordenar y adicionalmente un criterio de comparación

al → [7, 2, 5, 3, 2]

```
//main
final Comparator<Integer> PARES_IMPARES = (a,b) ->
{
    if(a%2==0 && b%2==1) return -1;
    if(a%2==1 && b%2==0) return 1;
    return 0;
};
System.out.println(al);
Collections.sort(al,PARES_IMPARES);
System.out.println(al);
Collections.sort(al);
System.out.println(al);
```

Primero los pares y luego los impares

Se ordenan en el orden natural de tipo del ArrayList

Salida consola

```
[7, 2, 5, 3, 2]
[2, 2, 7, 5, 3]
[2, 2, 3, 5, 7]
```

Métodos con ArrayList

Los ArrayList se pasan por referencia

Función que suma los valores de un ArrayList

```
static int sumar(ArrayList<Integer> al)
{
    int suma = 0;
    for(int v:al){
    }
        suma+=v;
    return suma;
}
```

Función que obtiene los valores pares contenidos en un ArrayList

```
static ArrayList pares(ArrayList<Integer> al)
{
    ArrayList<Integer> a = new ArrayList<Integer>();

    for(Integer v:al)
        if(v%2==0) a.add(v);
    return a;
}
```

Métodos con ArrayList (II)

Procedimiento que duplica los valores de un ArrayList

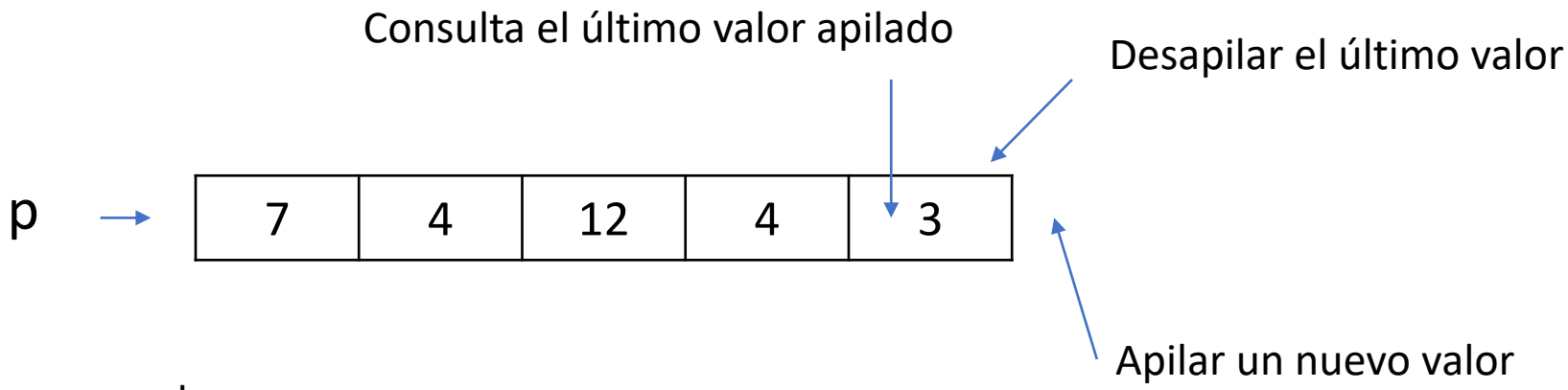
```
static void duplicar(ArrayList<Integer> al)
{
    for(int i=0;i<al.size();i++)
        al.set(i,al.get(i)*2);
}
```

Procedimiento que intercambia dos ArrayList

```
static void intercambiar(ArrayList<Integer> a, ArrayList<Integer> b)
{
    ArrayList<Integer> aux = a;
    a = b;
    b = aux;
}
```

Pilas

En Java se tiene la clase “Stack” definida en el paquete “java.util”, que permite apilar objetos de cualquier tipo.



- Mostrar por consola
- Comprobar si está vacío
- Clonar y vaciar
- Recorrer la pila

Métodos de **pilas**

Crear e inicializar una pila vacía

```
Stack<Integer> p = new Stack<Integer>();
```

Vaciar una pila

```
p.clear();
```

Apilar un valor “n”

```
p.push(n);
```

Desapilar el último valor apilado

```
p.pop();
```

Comprobar si una pila está vacía

```
p.isEmpty();
```

Clonar una pila

```
p1 = (Stack) p.clone();
```

Mostrar por consola una pila

```
Sytem.out.println(p);
```

Recorrer “al” iterando

```
for(Iterator<Integer> r = p.iterator(); r.hasNext();){
        n=r.next();
}
```

La clase “Iterator” está en el paquete “java.útil”

```
//main
Stack<Integer> p = new
Stack<Integer>();

p.push(in.leerInt("Entero: ",v->v>0));
p.push(in.leerInt("Entero: ",v->v>0));
p.push(in.leerInt("Entero: ",v->v>0));
p.push(in.leerInt("Entero: ",v->v>0));
System.out.println(p);
```

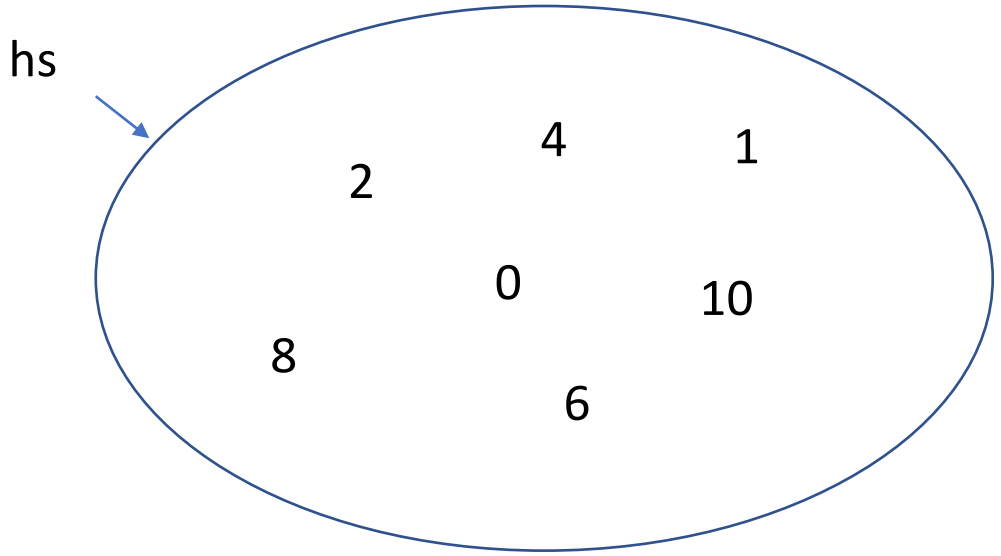
Salida consola

```
Entero: 8
Entero: 90
Entero: 11
[5, 8, 90, 11]
[7, 2, 5, 3, 2]
```

En los métodos las pilas se pasan por referencia

Conjunto hash

En Java se tiene una clase “HashtSet” definida en el paquete “java.util”, que permite almacenar objetos de cualquier tipo en un conjunto.



- Añadir un elemento si no existe
- Eliminar un valor si existe
- Obtener tamaño
- Comprobar si contiene un valor
- Comprobar si está vacío
- Clonar y vaciar
- Mostrar por consola
- Recorrer el conjunto

Métodos **conjuntos hash**

En los métodos los conjuntos se pasan por referencia

Crear e inicializar `HashSet<Integer> c = new HashSet<Integer>();`

Añadir el valor "n" si no existe `c.add(new Integer(a));` **Clonar** `c1 = c.clone();`

Vaciar `c.clear();`

Ver si está vacío `c.isEmpty();`

Tamaño `c.size(n)` → `int`

Eliminar el valor "n"

`c.remove(n);`

Comprobar si contiene el valor "n"

`c.contains(n)` → `boolean`

```
//main
HashSet<Integer> hs = new HashSet<Integer>();
for(int i=0; i<10;i++)
   hs.add((int)(Math.random()*10));
System.out.println(hs);
```

Salida consola

```
[1, 2, 3, 5, 7, 8, 9]
```

Aunque se generan 10 enteros, sólo aparecen 7 ya que se han repetido 3. Se muestran ordenados.

Mostrar por consola `Sytem.out.println(c);`

Recorrer iterando

```
for(Iterator<Integer> r = c.iterator(); r.hasNext();){
      n=r.next();
}
```

La clase "Iterator" está en el paquete "java.útil"

Tablas hash

En Java se tiene una clase “Hashtable” definida en el paquete “java.util”, que permite almacenar objetos de cualquier tipo a los cuales se les asocia un índice o clave de un tipo.

th →

CLAVE	VALOR
“Madrid”	23
“Roma”	100
“Paris”	67

- Vaciar una tabla hash
- Ver si un valor está contenido en la tabla hash
- Ver su una clave está contenida en la tabla hash
- Obtener el valor asociado a una clave
- Comprobar si la tabla hash está vacío
- Clonar una tabla hash
- Añadir un valor con una clave o reemplazarlo si ya existe para esa clave
- Eliminar el valor asociado a una clave
- Mostrar por consola una tabla hash
- Recorrer una tabla hash

Métodos de tablas hash

Crear e inicializar

```
Hashtable<String,Integer> th = new Hashtable<String,Integer>();
```

Vaciar `th.clear();` **Tamaño** `th.size()` → `int`

Ver si contiene el valor "n" `th.contains(n)` → `boolean`

Ver si contiene la clave "s" `th.containsKey(s)` → `boolean`

Obtener el valor asociado a la clave "s" `n = th.get(s);`

Ver si está vacía `th.isEmpty()` → `boolean`

Añadir el valor "n" con la clave "s" o reemplazar la clave "s" si ya existe

```
th.put(s, n); //Añadir
b = th.put(s, n); //Devuelve el valor "b" reemplazado
```

Eliminar el valor con la clave "s" `n = th.remove(s);`

Recorrer una tabla hash

Recorrer por las claves

```
for(Enumeration<String> e = th.keys(); e.hasMoreElements();)
{
        String clave = e.nextElement();
        int n = th.get(clave);
        …
}
```

La clase Enumeration está en el paquete java.util

Recorrer por los valores

```
for(Enumeration<Integer> e=th.elements();e.hasMoreElements();)
{
        int n = e.nextElement());
        …
}
```

En los métodos las tablas hash se pasan por referencia

Recorrer una **tabla hash** (II)

```
//main
Hashtable<String,Integer> th = new
Hashtable<String,Integer>();
for(int i=0; i<6;i++)
{
   String modulo = in.leerString("Modulo: ");
   int nota = in.leerInt("Nota: ",v->v>0 && v<=10);
   th.put(modulo,nota);
}

System.out.println(th);

int total = 0;
for(Enumeration<Integer>
e=th.elements();e.hasMoreElements();)
   total += e.nextElement();

System.out.printf("Media:
%1.1f\n",(double)total/th.size());
```

Salida consola

```
Modulo: BD
Nota: 7
Modulo: PRO
Nota: 6
Modulo: XML
Nota: 5
Modulo: FOL
Nota: 10
Modulo: ED
Nota: 7
Modulo: SI
Nota: 9
{XML=5, ED=7, SI=9, PRO=6, BD=7, FOL=10}
Media: 7,3
```

Programa que lee los módulos y notas de un alumno de DAM y obtiene su media