

# UT5

## Colecciones de tamaño fijo: arrays. Librería de clases: la clase String.

(1ª presentación)

Módulo – Programación (1º)

Ciclos – Desarrollo de Aplicaciones Multiplataforma | Desarrollo de Aplicaciones Web

CI María Ana Sanz

---

# Contenidos

- Arrays en Java
  - declarar un array
  - crear un array
  - dar valores a un array
  - arrays como parámetros
  - arrays como valor de retorno
  - arrays como atributos
  - arrays de objetos
  - arrays bidimensionales
- Operaciones con arrays
  - búsqueda en arrays
  - ordenación de arrays
- La API de Java
  - documentación de una clase
- La clase String
- La clase StringBuilder
- La clase Scanner como *tokenizer*
- Signatura del método *main()*
- El tipo enum

# Objetivos

- Comprender la necesidad de los arrays
- Aprender a
  - declarar y crear un objeto array (unidimensional y bidimensional)
  - dar valores un array
  - procesar un array
  - buscar un valor dentro de un array y ordenar el array
  - trabajar con arrays de objetos
- Manejar correctamente la API de Java, en particular, las clases
  - String, StringBuilder, Scanner
- Entender la signatura del método *main()*

# Necesidad de los arrays

- Agrupar una serie de valores para ser tratados de forma homogénea

```
public class Equipo
{
    private Jugador jugador1;
    private Jugador jugador2;
    private Jugador jugador3;
    .....
}
```

¿Y si queremos guardar los 22 jugadores de un equipo?

```
public class Alumno
{
    private String nombre;
    private int nota1;
    private int nota2;
    private int nota3;
    .....
}
```

¿Y si queremos guardar las notas de 10 asignaturas?

# Necesidad de los arrays

- Podemos agrupar los valores creando una estructura de datos de tamaño fijo: el **array**
  - **Estructura de datos**
    - forma de organizar un conjunto de datos relacionados para facilitar su manipulación
- Otros ejemplos
  - relación de canciones de un CD
  - frecuencia de aparición de cada una de las caras de un dado en una serie de lanzamientos
  - nº de días que tiene cada uno de los meses del año

# ¿Qué es un array?

- **Colección de valores**, **todos del mismo tipo**, que se tratan de una manera específica.
- Físicamente los elementos de un array se sitúan en memoria uno detrás de otro.
- Los elementos del array se **identifican por la posición** que ocupan dentro de él.
- Los valores que puede almacenar un array son:
  - tipo primitivo – int, float, double, char, boolean, .....
  - tipo referencia – array de objetos

79	87	94	82	67	98	87	81	74	91
0	1	2	3	4	5	6	7	8	9

vista lógica de un array

# Declarar un array

- **tipo[] nombre\_del\_array**

- `int[ ] arrayEnteros;`
- `char[ ] arrayCaracteres;`
- `float [ ] notas;`
- `Jugador[] equipo;`

notas

null

se declara una variable  
referencia, no se establece  
el tamaño ni el valor

- En la declaración no se especifica nada acerca del tamaño del array (no se reserva memoria para él).

- Ejer 5.1

## Ejer 5.1

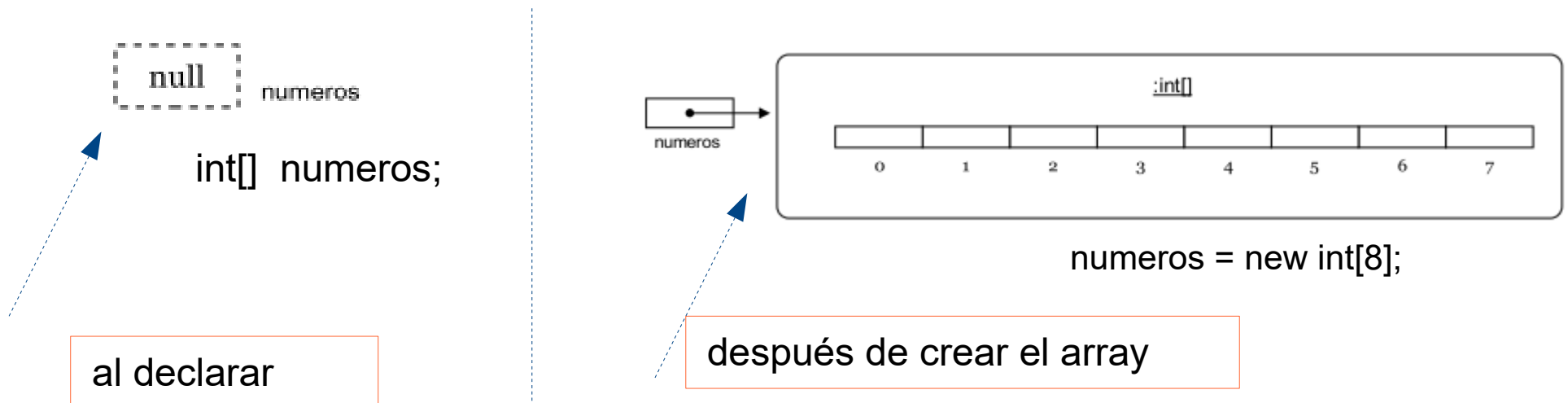
---

- `double[] precios;`
- `boolean[] plazasParking;`
- `String[] nombres;`
- `double[] lluviaNoviembre;`



# Creando objetos arrays

- Un array es un objeto
- Se construye de forma especial a través del operador **new()**
  - no se llama al constructor con nombre de clase
  - **new tipo[expresión\_entera];**
- **int[ ] numeros;** // declaración del array
- **numeros = new int[8];** //creación del array



# Creando objetos arrays

- Declarar y crear en la misma sentencia
  - **double[ ] precios = new double[10];**
- Una vez se ha creado un array su tamaño no puede ser modificado.
- Java asigna a los elementos de un array un valor por defecto
  - cero si son enteros o reales
  - '\u0000' para caracteres
  - false para valores lógicos
  - null para los objetos.
- Ejer 5.2

## Ejer 5.2

---

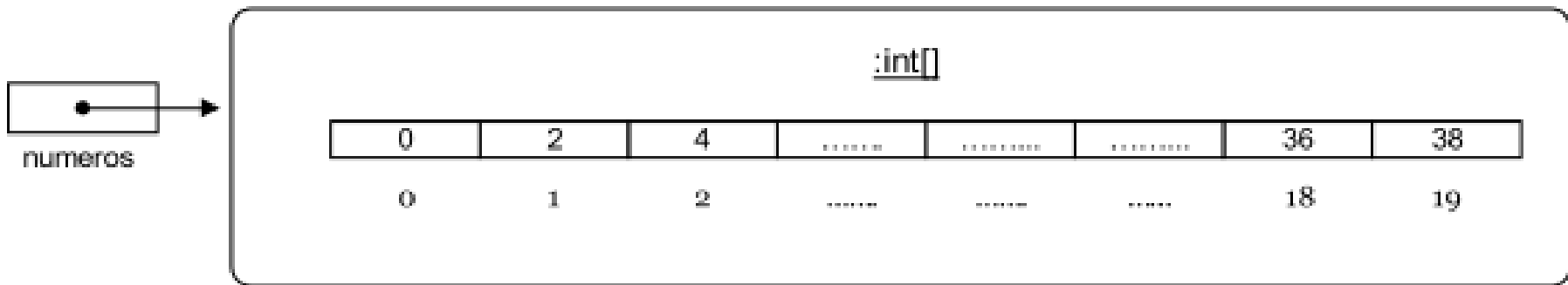
- `precios = new double[20];`
- `plazasParking = new boolean[n];`
- `nombres = new String[MAX];`
- `lluviaNoviembre = new double[30];`

# Acceso a los elementos de un array

- Atributo público – **length** (no es un método)
  - indica la longitud del array (nº elementos reservados para él)
- `int[ ] numeros = new int[20];`
  - `numeros.length` devuelve el valor 20
- Para acceder a los elementos de un array
  - indicar la posición o índice que ocupa el elemento dentro de él
  - el valor del índice varía entre **0 y length – 1**
  - si índice fuera del rango se genera un error (una excepción)  
**ArrayIndexOutOfBoundsException.**

# Acceso a los elementos de un array

```
int[ ]  numeros = new int[20];  
.....  
for (int i = 0; i < numeros.length; i++) {  
    numeros[i] = 2 * i;  
}
```



- Cada uno de los elementos de un array puede ser tratado como cualquier otra variable.
  - `numeros[2]` es una variable cuyo valor es 4
  - podemos operar con ella como con otra variable cualquiera

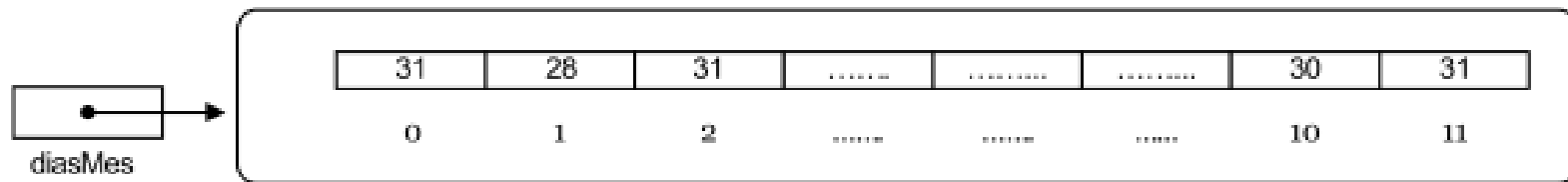
# Declarar e inicializar un array

- Si sabemos de antemano los valores que tendrá un array podemos

- declarar e inicializar en un paso

- **int[ ] diasMes = {31, 28, 31, 30, ....., 30, 31};**

no se indica tamaño



- **int[ ] diasMes;**
- **diasMes = {31, 28, 31, 30, ....., 30, 31};**

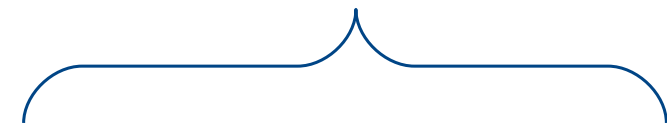
INCORRECTO

- **int[ ] diasMes;**
- **diasMes = new int[]{31, 28, 31, 30, ....., 30, 31};**

CORRECTO

# Declarar e inicializar un array

- `private static final String[] NOMBRE_MESES =  
    {"Enero", "Febrero", ....., "Diciembre"};`
- `private static final String[] DIAS_SEMANA =  
    {"Lunes", "Martes", ....., "Domingo"};`
- `Poligono triangulo = new Poligono(new int[]{0, 10, 5},  
    new int[]{10, 20, 15});`



arrays anónimos

## Ejer 5.3 / 5.4

- Ejer 5.3

- `int numeroElementos = diasMes.length;`
- `diasMes[1] = 29;`

- Ejer 5.4

```
public void escribirArray()
{
    int[] numeros = {1,2,3,4,5,6,7,8,9,10};
    for (int i = 0; i < numeros.length; i++) {
        System.out.println(i + "\t" + numeros[i]);
    }
}
```



# Ejer 5.5

## ■ Ejer 5.5

```
/**
 * @param diaSemana valor entre 1 y 7
 */
public String encontrarNombreDia(int diaSemana)
{
    final String[] NOMBRES = {"Lunes", "Martes", ....., "Domingo"};
    if (diaSemana >= 1 && diaSemana <= 7) {
        return nombres[diaSemana - 1];
    }
    return "Día Incorrecto";
}
```

# Arrays como argumentos

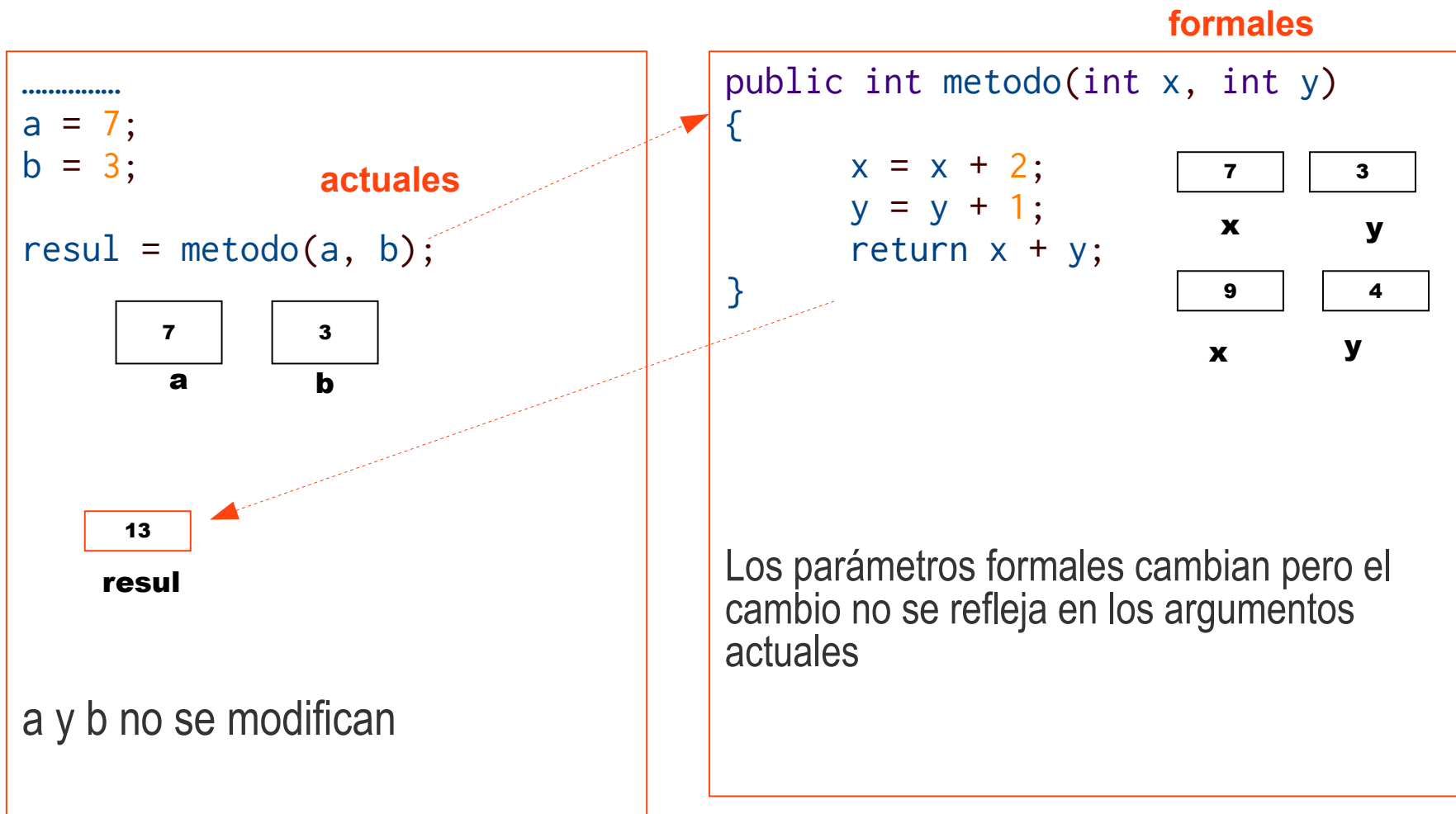
- Un array puede ser pasado como argumento a un método
- Se pasa la **referencia al array**, el **nombre** del array
  - si el **método modifica el parámetro** los cambios se reflejarán en el **array original**

```
public void invertir(int[] valores)
{
    int limiteDerecha = valores.length - 1;
    for (int i = 0; i < (valores.length / 2); i++) {
        int aux = valores[i];
        valores[i] = valores[limiteDerecha];
        valores[limiteDerecha] = aux;
        limiteDerecha--;
    }
}
```

```
public void escribir(int[] valores)
{
    for (int i = 0; i < valores.length; i++) {
        System.out.print(valores[i] + ",");
    }
    System.out.println();
}
```

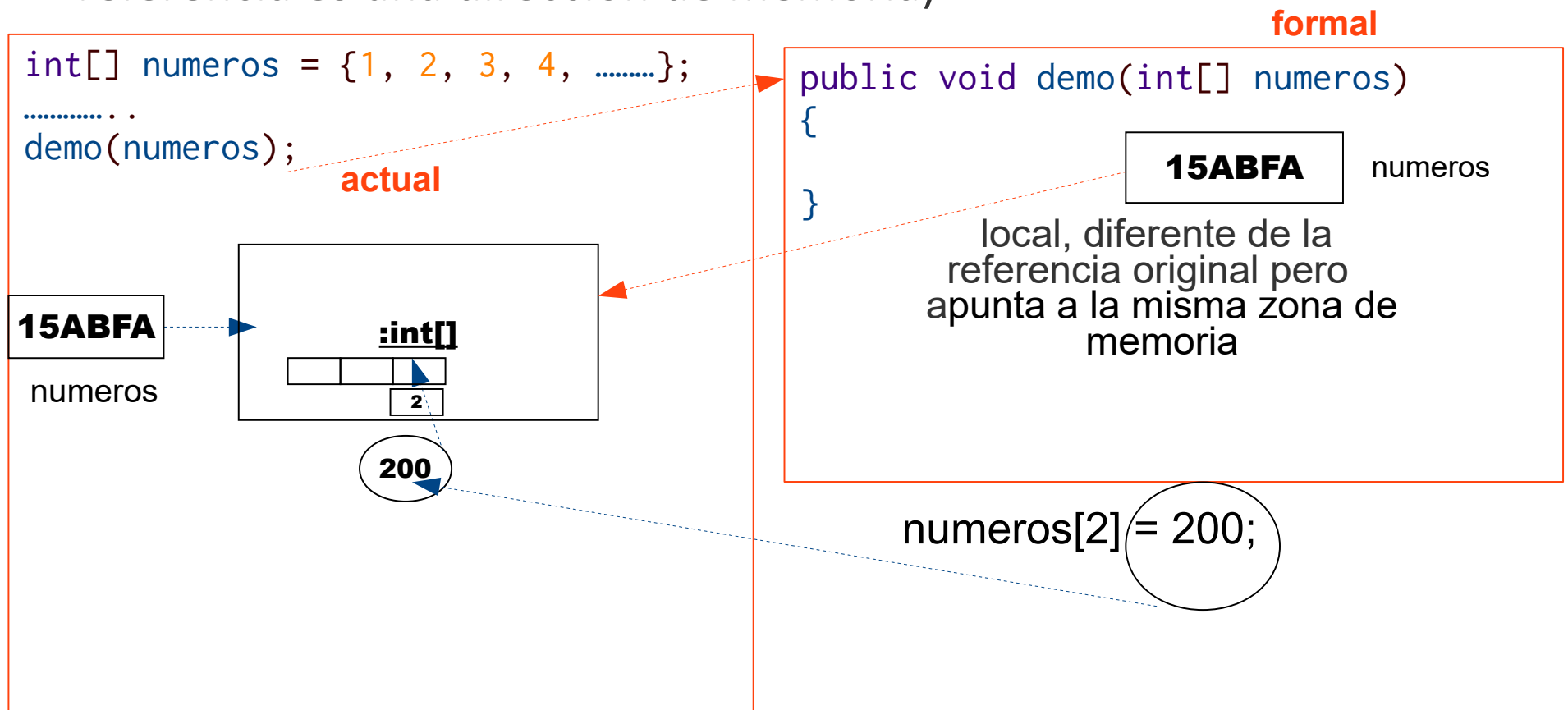
# Paso de parámetros por valor

- En Java el paso de parámetros es **por valor**



# Paso de parámetros por valor (paso de arrays)

- Cuando se pasa un array se pasa la **referencia al array** (una referencia es una dirección de memoria)



# Array como valor de retorno

- Un método puede devolver a través de la sentencia return un array

```
public int[ ] invertir(int[] valores)
{
    int[ ] resultado = new int[valores.length];
    for (int i = 0; i < valores.length ; i++) {
        resultado[i] = valores[valores.length - i - 1];
    }
    return resultado;
}
```

este array no cambia

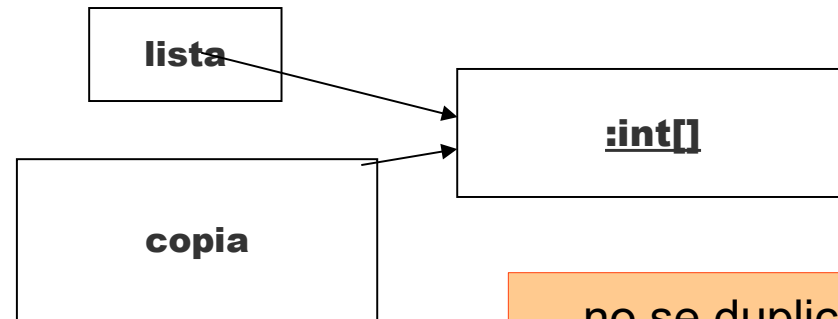
creamos un nuevo array

lo devolvemos como resultado

- Ejer 5.6 a 5.11

# Duplicar un array

```
public int[] duplicar(int[] lista)
{
    int[] copia;
    copia = lista;
    return copia;
}
```



no se duplica el array

- Para duplicar:
  - crear un nuevo array y
    - copiar elementos individuales uno a uno o
    - utilizar **System.arraycopy()**
  - utilizar **Arrays.copyOf()**
  - usar **clone()**

# System.arraycopy()

- Método estático de la clase System
  - System en el paquete *java.lang*
- **public static void  
arraycopy(int[] fuente, int posFuente,  
int[] destino, int posDestino, int longitud)**
  - System.arraycopy(lista, 0, copia, 0, lista.length);
  - copia se crea previamente con new int[lista.length]

```
public int[] duplicar(int[] lista)
{
    int[] copia = new int[lista.length];
    System.arraycopy(lista, 0, copia, 0, lista.length);
    return copia;
}
```

# Arrays.copyOf()

- Método estático de la clase Arrays
  - Arrays en el paquete *java.util*

<https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Arrays.html>

- **public static int[] copyOf(int[] fuente, int nuevaLongitud)**

- **fuente** – el array a ser copiado
- **nuevaLongitud** – la longitud del array copiado que se devuelve
- se devuelve una copia de fuente truncada o rellenada con 0, null, ... hasta obtener la longitud especificada

El tamaño de la copia puede ser menor, igual o mayor que el tamaño del que va a ser copiado.

- `int[] original = {42, 55, 21};`
- `int[] copia = Arrays.copyOf(original, original.length);`
- no hay que crear el array copia previamente



# Arrays como atributos

```
public class EjemploArray
{
    private int[ ] números;
    public EjemploArray()
    {
        numeros = new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9};
    }
    .....
}
```

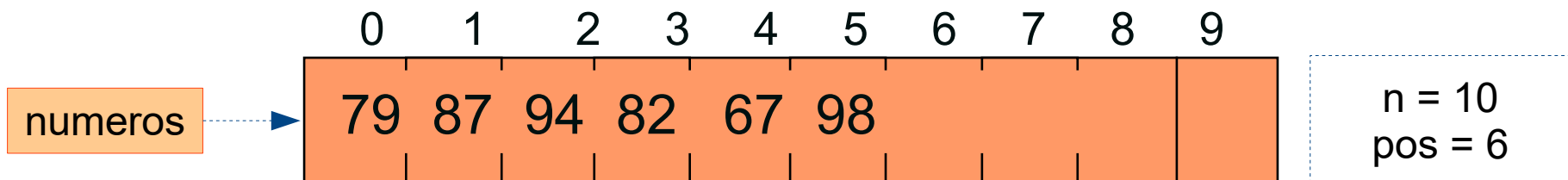
```
public class EjemploArray
{
    private int[ ] números;
    public EjemploArray()
    {
        numeros = new int[9];
        inicializar(numeros);
    }
    private void inicializar(int[ ] numeros)
    {
        for (int i = 0; i < numeros.length;
            i++) {
            numeros[i] = i + 1;
        }
    }
}
```

# Arrays como atributos

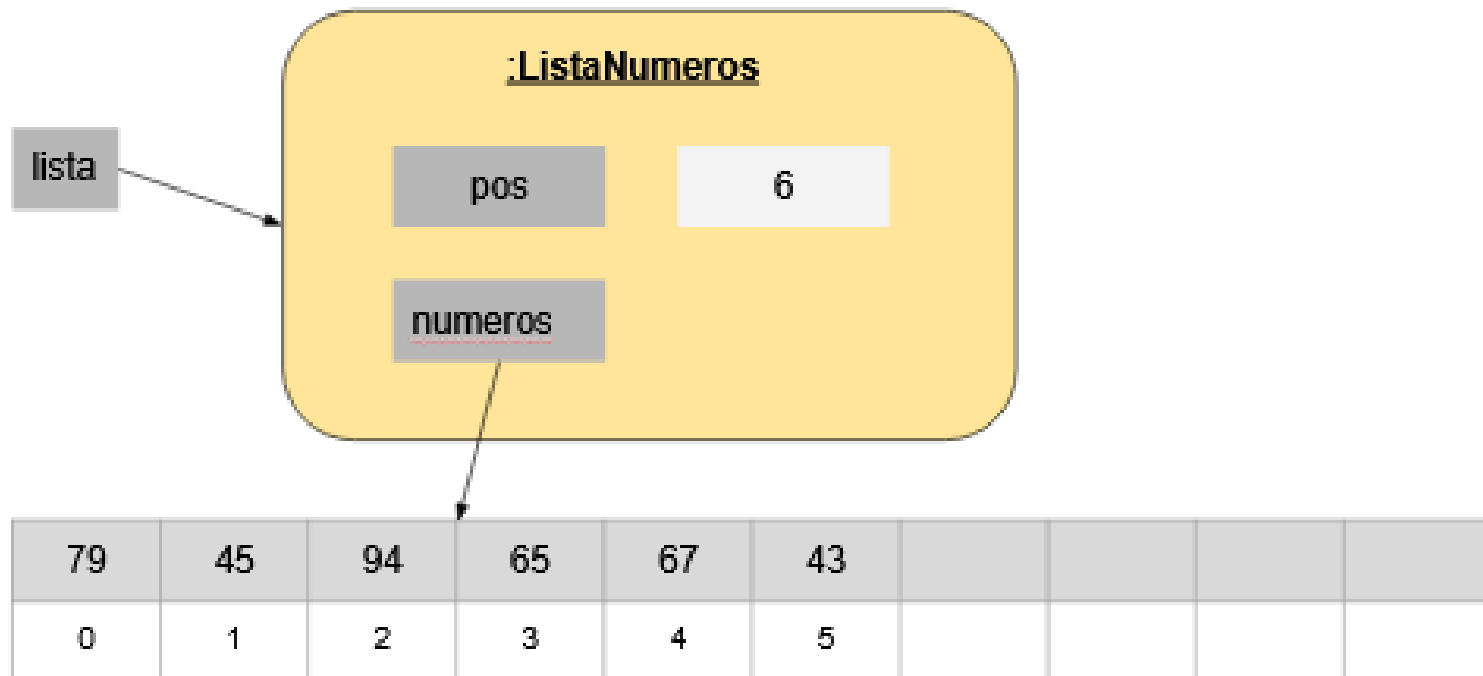
```
public class ListaNumeros
{
    private int[] numeros;
    private int pos;
    public ListaNumeros(int n)
    {
        numeros = new int[n];
        pos = 0; //podría ser -1
    }
    public ListaNumeros(int[] array)
    {
        numeros = array;
        pos = array.length;
    }
}
```

```
public void addNumero(int valor)
{
    if (pos < numeros.length) {
        numeros[pos] = valor;
        pos++;
    }
}
```

**pos** – nº real de elementos en el array –  
la longitud lógica  
**n** – tamaño máximo del array – longitud física



# Arrays como atributos



# Arrays como atributos

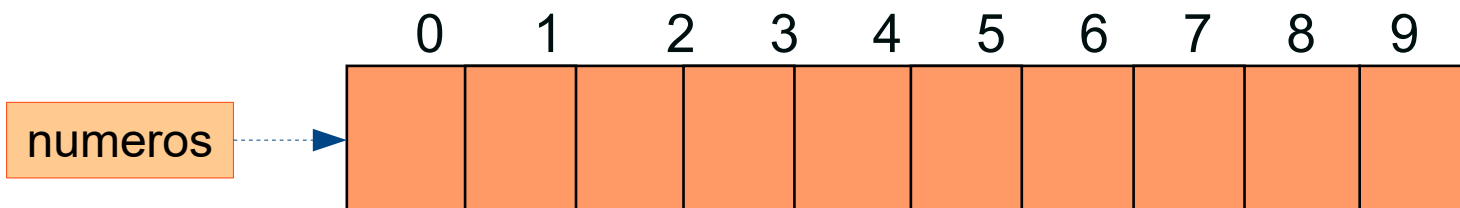
```
public class ListaNumeros
{
    private int[] numeros;
    private int pos;

    public ListaNumeros(int n)
    {
        numeros = new int[n];
        pos = 0; //podría ser -1
    }
}
```

```
public void addNumero(int valor)
{
    if (pos < numeros.length) {
        numeros[pos] = valor;
        pos++;
    }
}
```

n = 10  
pos = 0

Inicialmente al hacer new ListaNumeros(10)



# Arrays como atributos

```
public class ListaNumeros
{
    private int[] numeros;
    private int pos;

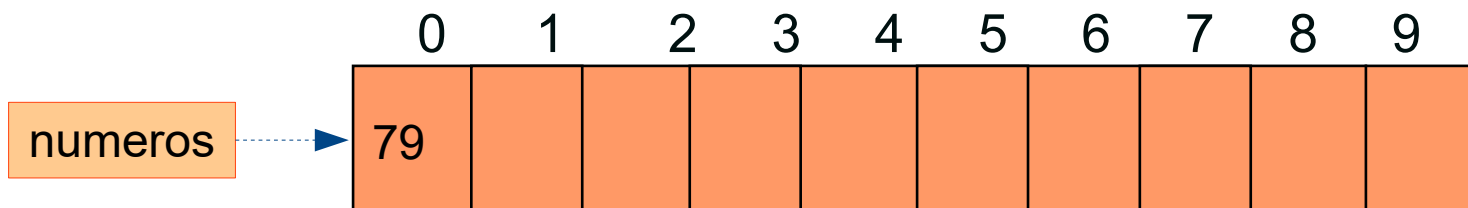
    public ListaNumeros(int n)
    {
        numeros = new int[n];
        pos = 0; //podría ser -1
    }
}
```

```
public void addNumero(int valor)
{
    if (pos < numeros.length){
        numeros[pos] = valor;
        pos++;
    }
}
```

**n = 10**  
**pos = 0**

**addNumero(79)**

**n = 10**  
**pos = 1**



# Arrays como atributos

```
public class ListaNumeros
{
    private int[] numeros;
    private int pos;

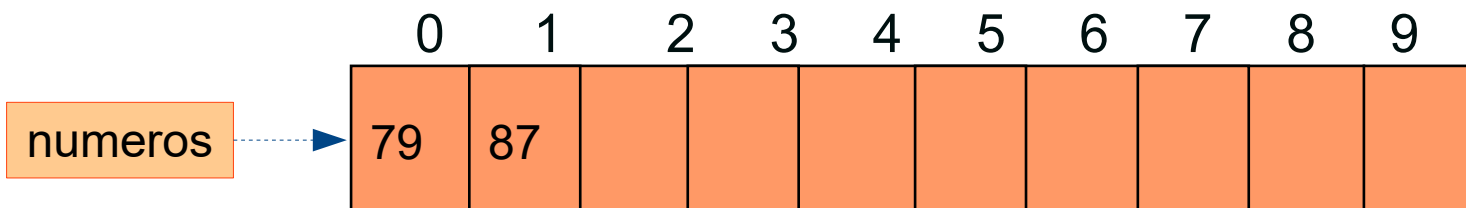
    public ListaNumeros(int n)
    {
        numeros = new int[n];
        pos = 0; //podría ser -1
    }
}
```

```
public void addNumero(int valor)
{
    if (pos < numeros.length){
        numeros[pos] = valor;
        pos++;
    }
}
```

**n = 10**  
**pos = 1**

**addNumero(87)**

**n = 10**  
**pos = 2**



# Arrays como atributos

```
public class ListaNumeros
{
    private int[] numeros;
    private int pos;

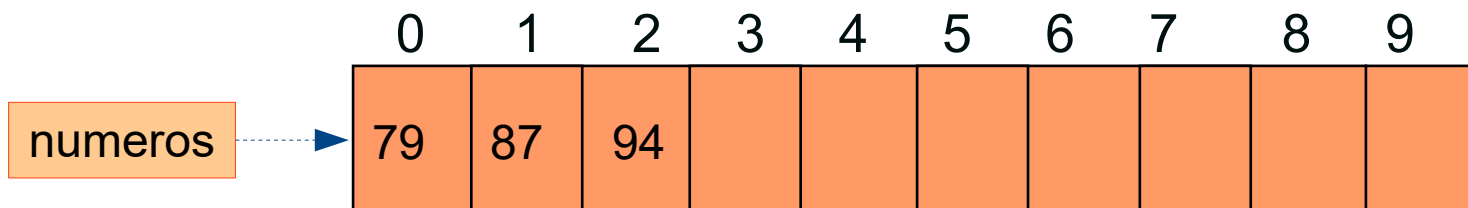
    public ListaNumeros(int n)
    {
        numeros = new int[n];
        pos = 0; //podría ser -1
    }
}
```

```
public void addNumero(int valor)
{
    if (pos < numeros.length){
        numeros[pos] = valor;
        pos++;
    }
}
```

**n = 10**  
**pos = 2**

**addNumero(94)**

**n = 10**  
**pos = 3**



# Arrays como atributos

```
public class Alumno
{
    private static final int MAX_ASIGNATURAS = 10;
    private String nombre;
    private int[] notas;
    private int pos;
    public Alumno(String nombre)
    {
        this.nombre = nombre;
        notas = new int[MAX_ASIGNATURAS];
        pos = 0;
    }
    public void registrarNota(int nota)
    {
        if (pos < notas.length) {
            notas[pos] = nota;
            pos++;
        }
    }
}
```

■ Ejer 5.12 y 5.13



# Ejercicios adicionales

---

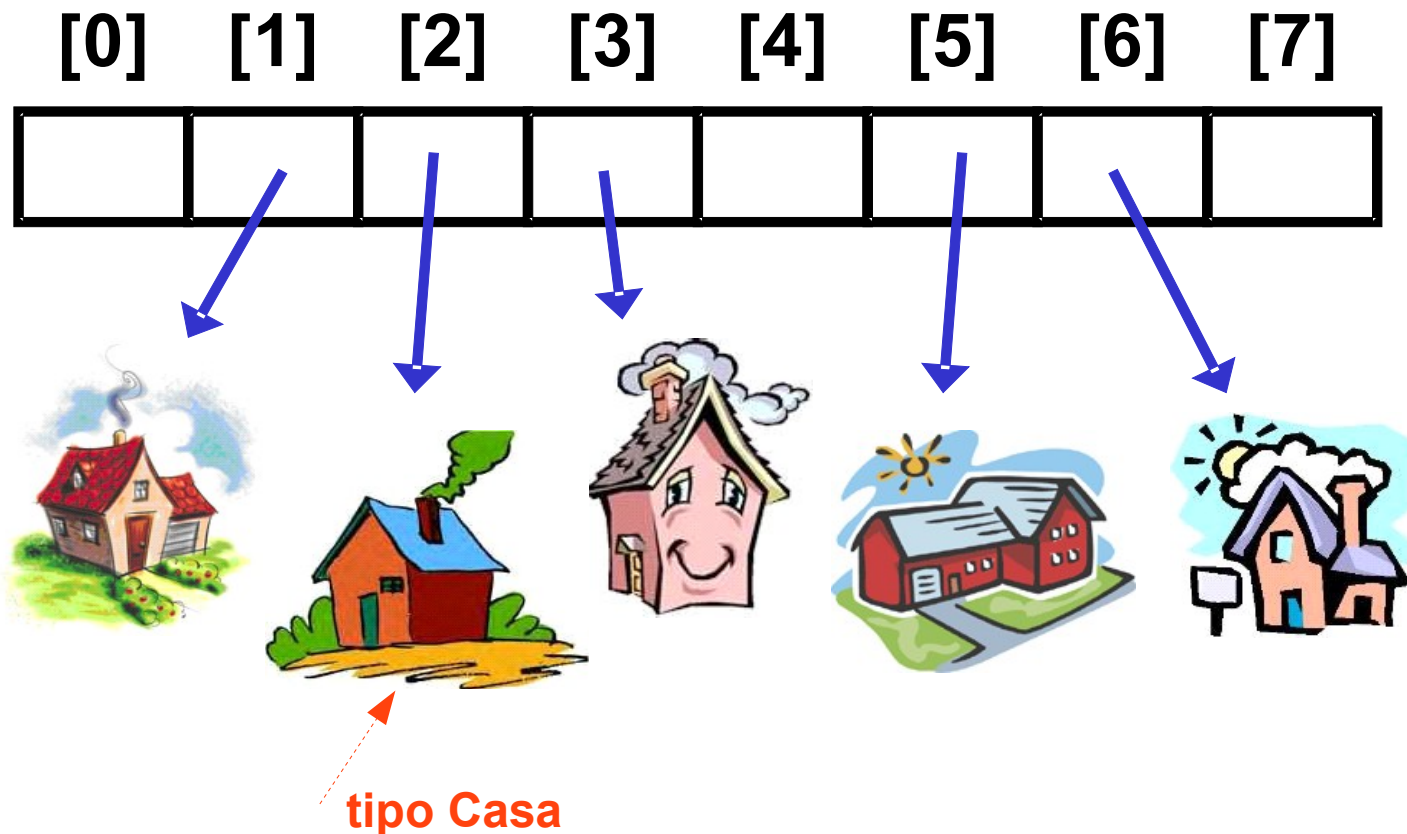
EJAD01 Histograma

EJAD02 Frecuencia dado

EJAD03 Calculadora números

# Arrays de objetos

- Arrays con **tipo base** de los elementos **un tipo objeto** (tipo referencia)

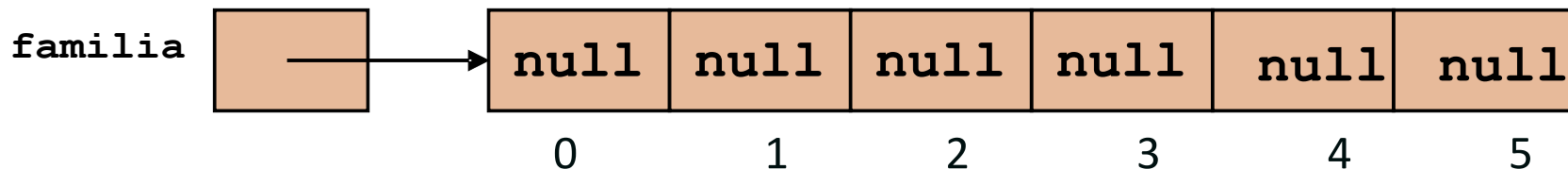


# Arrays de objetos

- `String[] listaPalabras = new String[10];`
- `Persona[] familia = new Persona[6];`
- `Alumno[] curso = new Alumno[MAX];`
- Cuando se declara el array de objetos Persona
  - `Persona[] familia;`  

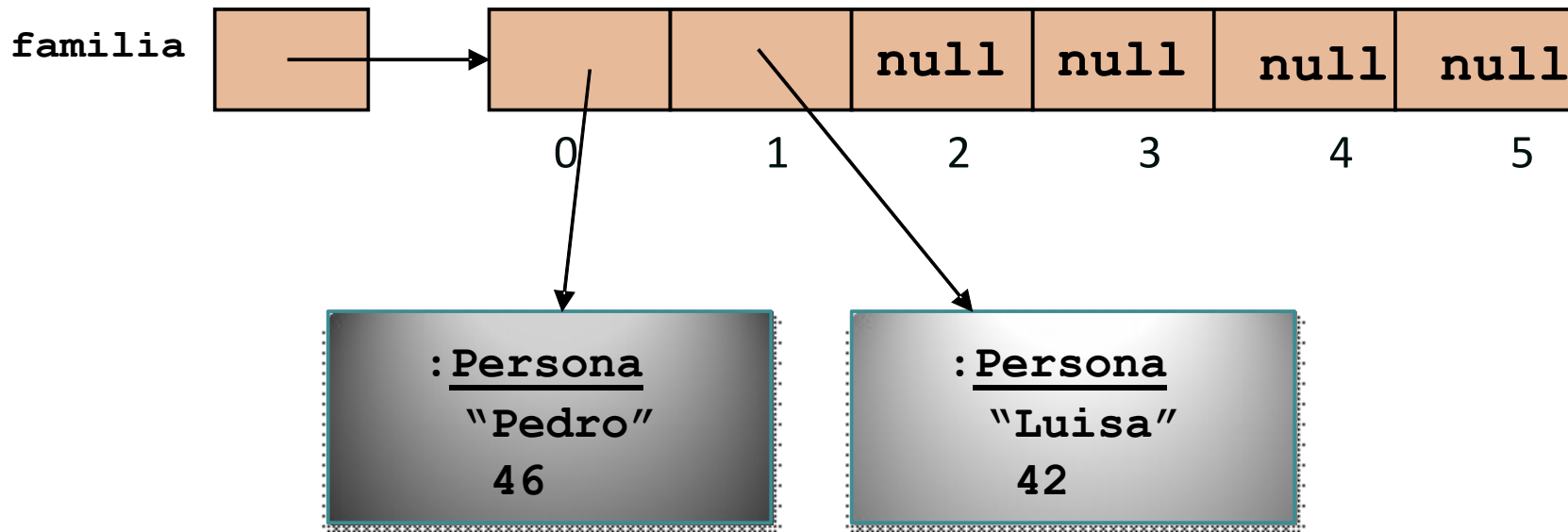
`familia`

`null`
- Cuando se crea el array de objetos Persona
  - `Persona[] familia = new Persona[6];`



# Arrays de objetos

- Cuando se crean objetos de tipo Persona y se asignan a elementos del array
  - familia[0] = new Persona("Pedro", 46);
  - familia[1] = new Persona("Luisa", 42);



# Ejer 5.14

---

## Ejercicios

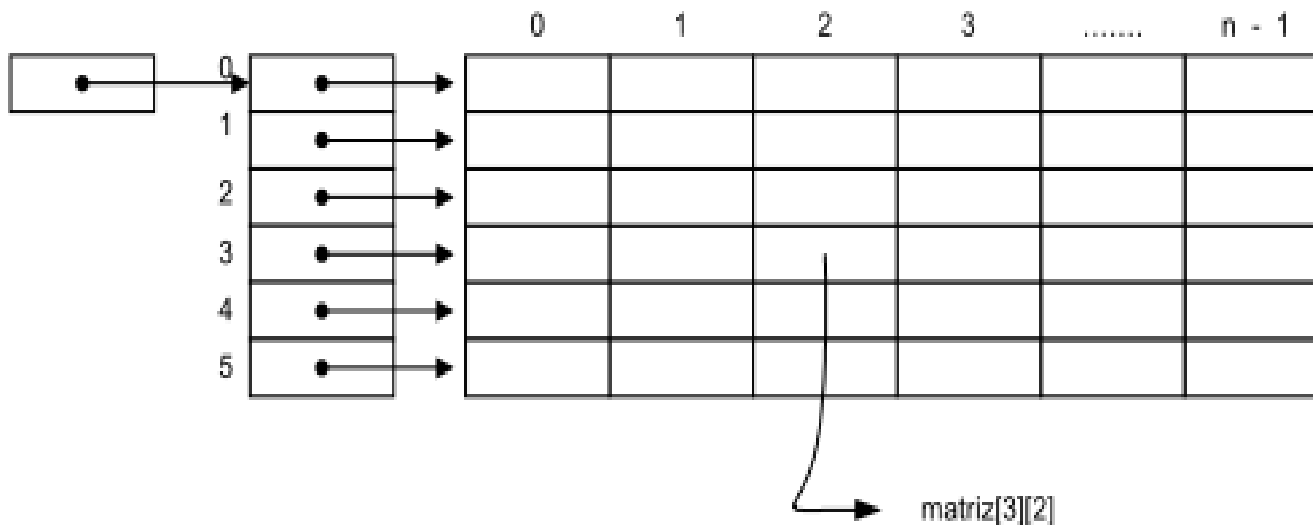
# Arrays de dos dimensiones

- Representación tabular , filas y columnas

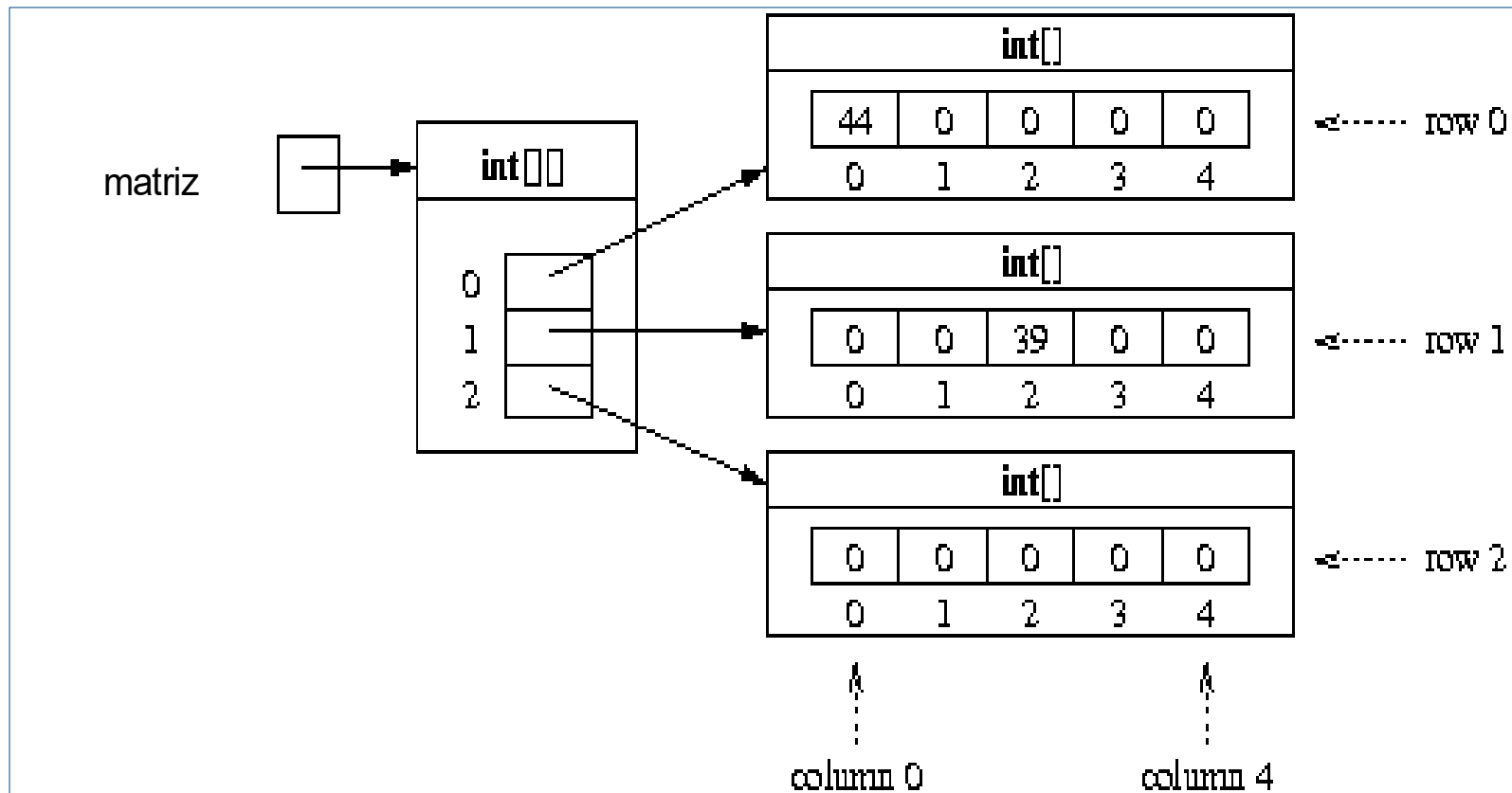


# Arrays de dos dimensiones

- Un array bidimensional es un **array de arrays**
- **tipo[][] nombre\_array**
  - `int[][] matriz; // declarar el array`
  - `matriz = new int[6][5]; // crear array de 6 filas y 5 columnas`
  - `matriz[3][2] = 16; // asignar un valor`



# Arrays de dos dimensiones





# Acceso a elementos en un array 2D

- Indicar 2 índices: fila y columna
  - `matriz[2][4]` // primero siempre fila, segundo siempre columna
- **`matriz.length`** – nº de filas del array
- **`matriz[fila].length`** – nº elementos del array `matriz[fila]`

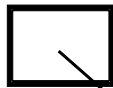
```
for (int fila = 0; fila < matriz.length; fila++) {  
    for (int columna = 0; columna < matriz[fila].length; columna++) {  
        matriz[fila][columna] = (int) (Math.random() * 30 + 1);  
    }  
}
```

# Declarar e inicializar un array 2D en un paso

```
int[] [] miArray = { {1, 2, 3, 4},  
                     {5, 6, 7, 8},  
                     {9, 10, 11, 12},  
                     {0, 6, 8, 18} };
```

```
int[][] miArray;  
miArray = new int[][] { {1, 2, 3, 4},  
                        {5, 6, 7, 8},  
                        {9, 10, 11, 12},  
                        {0, 6, 8, 18} };
```

# Arrays con filas de distinta longitud (ragged arrays)

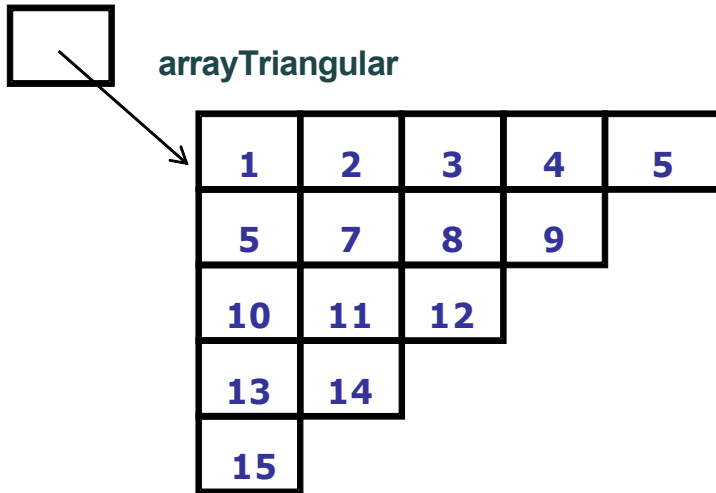


arrayTriangular

1	2	3	4	5
5	7	8	9	
10	11	12		
13	14			
15				

```
int[] [] arrayTriangular = { {1, 2, 3, 4, 5},  
                              {5, 7, 8, 9},  
                              {10, 11, 12},  
                              {13, 14},  
                              {15}  };
```

# Arrays con filas de distinta longitud (ragged arrays)



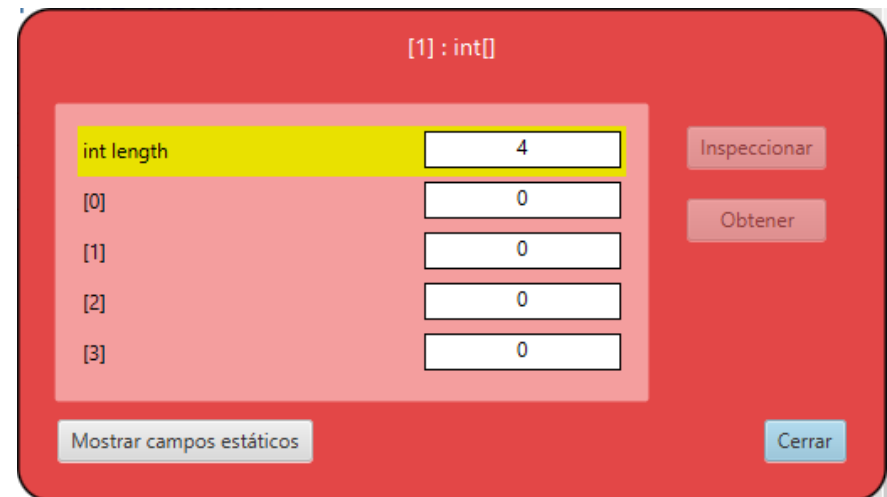
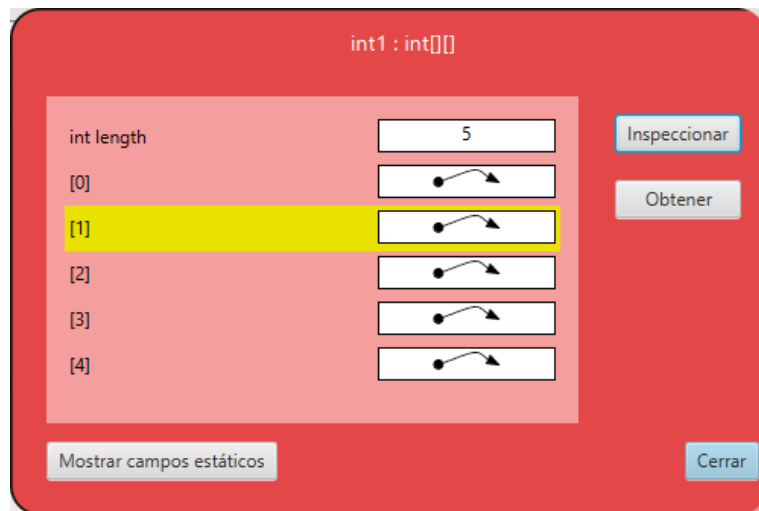
```
int[][] arrayTriangular = new int[5][];
```

```
arrayTriangular[0] = new int[] {1, 2, 3, 4, 5};  
arrayTriangular[1] = new int[] {5, 7, 8, 9};  
arrayTriangular[2] = new int[] {10, 11, 12};  
arrayTriangular[3] = new int[] {13, 14};  
arrayTriangular[4] = new int[] {15};
```

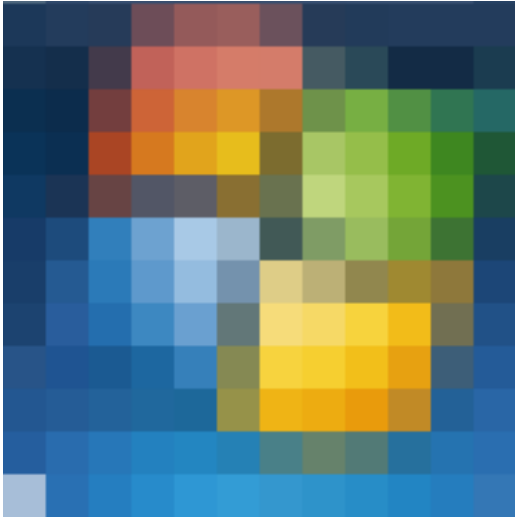
# Arrays con filas de distinta longitud (ragged arrays)

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

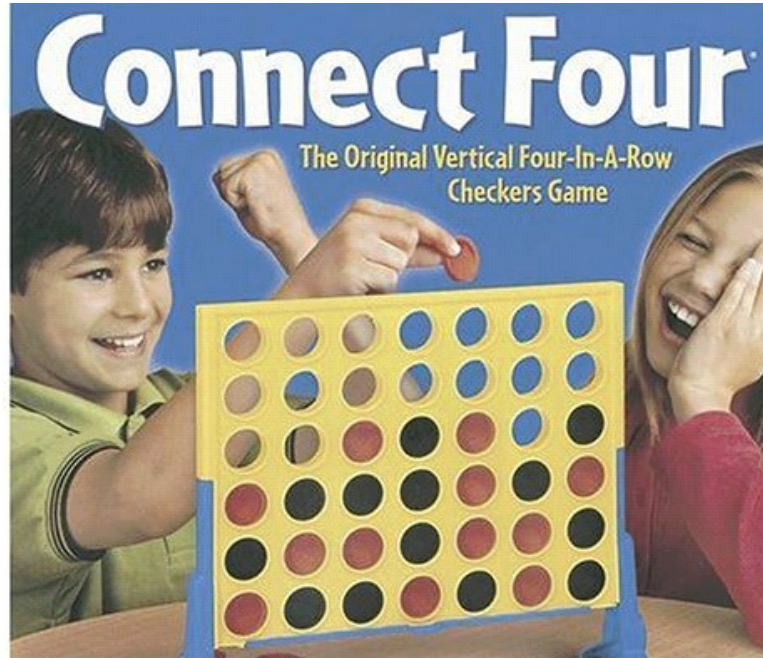
```
arrayTriangular[0] = new int[5];  
arrayTriangular[1] = new int[4];  
arrayTriangular[2] = new int[3];  
arrayTriangular[3] = new int[2];  
arrayTriangular[4] = new int[1];
```



# Arrays 2D - Aplicaciones



Imágenes – array 2d como conjunto de pixels



Juegos de tablero

- Ejer 5.15, 5.16 y 5.17

# Ejercicios

---

## Ejercicios adicionales

# Ejercicios adicionales

---

Clase Arrays2D (Moodle)



# Borrar un elemento de un array

- `public void borrarElemento(int posicion)`
  - borrar el elemento que está en *posicion*
  - asumimos *posicion* correcta

6	13	14	25	33	43	51	53	72	84		
0	1	2	3	4	5	6	7	8	9	10	11

pos = 10, nº real  
elementos de

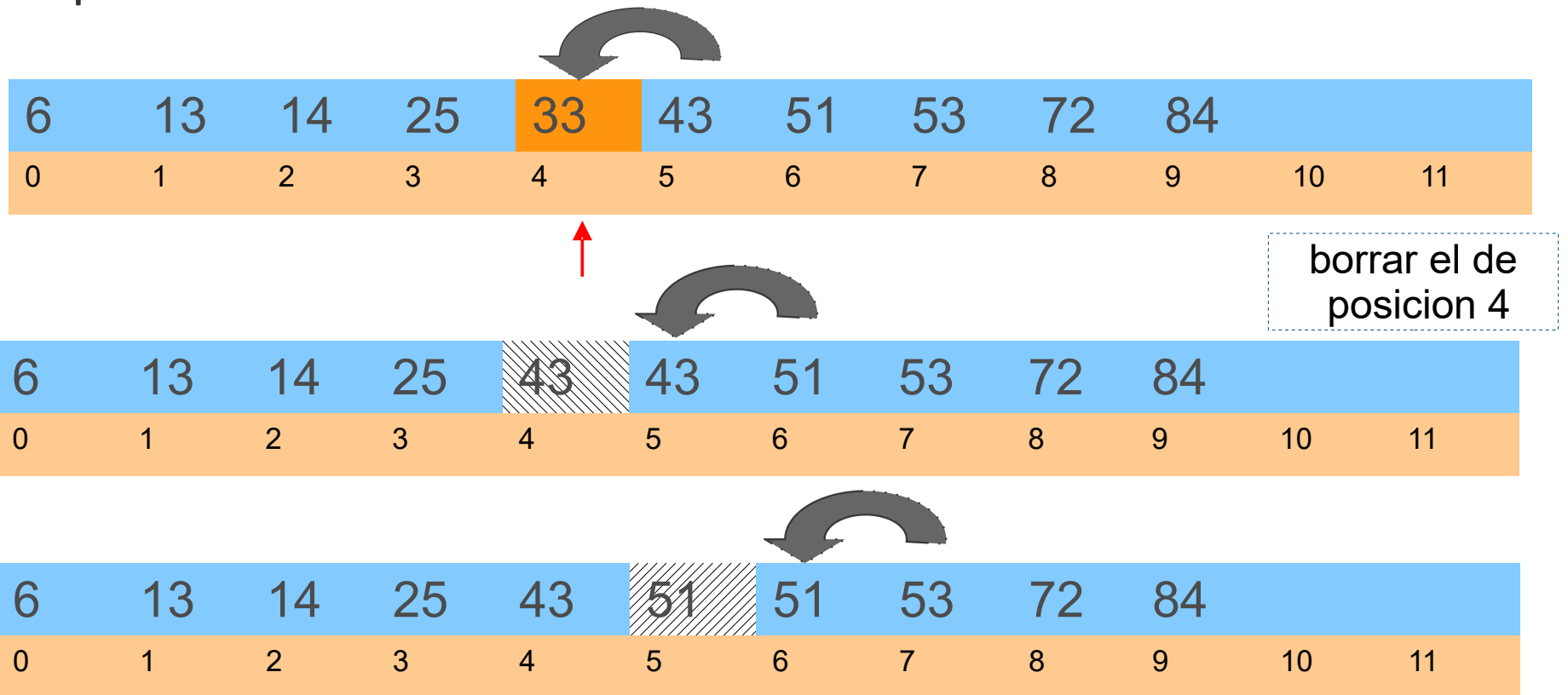


posicion

borrar el de  
posicion 4

# Borrar un elemento de un array

- Se desplazan todos los que están a la derecha una posición a la izquierda



# Borrar un elemento de un array

- Decrementar el nº de elementos reales del array
  - **pos--**

6	13	14	25	43	51	53	72	84	84		
0	1	2	3	4	5	6	7	8	9	10	11



valor actual de  
pos = 9

# Borrar un elemento de un array

```
public class ListaNumeros
{
    private int[] numeros;
    private int pos;
    /**
     * Borrar un elemento de una posición p
     */
    public void borrar(int p)
    {
        if (p >= 0 && p < pos) { // posición correcta
            // desplazar a la izquierda
            for (int i = p + 1; i < pos; i++) {
                numeros[i - 1] = numeros[i];
            }
            pos--;
        }
    }
}
```

# Borrar un elemento de un array (usando `System.arraycopy()` )

```
public class ListaNumeros
{
    private int[] numeros;
    private int pos;

    /**
     * Borrar un elemento de una posición p
     */
    public void borrar(int p)
    {
        if (p >= 0 && p < pos) { // posición correcta

            System.arraycopy(numeros, p + 1,
                             numeros, p, pos - p - 1);

            pos--;
        }
    }
}
```

el mismo efecto que  
el desplazamiento a la izquierda

# Borrar varios elementos de un array -

Ej. Borrar todos los impares

```
/**
 * Borrar varios elementos
 */
public void borrarImpares()
{
    for (int i = 0; i < pos; i++)
        if (numeros[i] % 2 != 0) {
            borrar(i);
        }
}
```



**INCORRECTO**

## No con for hacia adelante

Al borrar un impar de una posición desplazamos a la izquierda, en esa misma posición puede aparecer otro impar.

# Borrar varios elementos de un array -

Ej. Borrar todos los impares

```
/**
 * Borrar varios elementos
 */
public void borrarImpares()
{
    int i = 0;
    while (i < pos) {
        if (numeros[i] % 2 != 0) {
            borrar(i);
        }
        else {
            i++;
        }
    }
}
```

**CORRECTO**

## Con un while

Al borrar un impar de una posición desplazamos a la izquierda, en esa misma posición puede aparecer otro impar.

# Borrar varios elementos de un array -

Ej. Borrar todos los impares

```
/**
 * Borrar varios elementos
 */
public void borrarImpares()
{
    for (int i = pos - 1; i >= 0; i--) {
        if (numeros[i] % 2 != 0) {
            borrar(i);
        }
    }
}
```

**CORRECTO**

DESPLAZAR A LA DERECHA

**Con for empezando desde el final**

Al borrar un impar de una posición empezamos a la izquierda, en esa misma posición puede aparecer otro impar.

DESPLAZAR A LA IZQUIERDA  
Empezar desde 0 hasta pos-1

DESPLAZAR A LA IZQUIERDA  
Empezar desde 0 hasta pos-1  
n[i] = n[i + 1]  
No hay problemas con los valores de las variables

SI	NO	SI	NO
n[2] = n[3]	n[4] = n[5]	n[2] = n[3]	n[4] = n[5]
n[3] = n[4]	n[3] = n[4]	n[3] = n[4]	n[3] = n[4]
n[4] = n[5]	n[2] = n[3]	n[4] = n[5]	n[2] = n[3]



# Búsqueda en arrays

- Operación habitual en un array
  - buscar un elemento dentro de él
    - ej. buscar una determinada palabra en una lista de palabras, localizar un determinado título de canción en un CD, ....
- Algoritmos de búsqueda más comunes
  - **Búsqueda lineal**
  - **Búsqueda dicotómica**

## DESPLAZAR A LA IZQUIERDA

Empezar desde 0 hasta pos - 1

$n[i] = n[i + 1]$

No hay problemas con los valores de las variables

SI

$n[2] = n[3]$

$n[3] = n[4]$

$n[4] = n[5]$

for (int i = 0; i < pos; i++){  
   $n[i] = n[i + 1]$ }

NO

$n[4] = n[5]$

$n[3] = n[4]$

$n[2] = n[3]$

todos valdrian  $n[5]$



## DESPLAZAR A LA DERECHA

Empezar desde pos - 1 hasta 0

$n[i] = n[i - 1]$

No hay problemas con los valores de las variables

NO

$n[1] = n[0]$

$n[2] = n[1]$

$n[3] = n[2]$

SI

$n[3] = n[2]$

$n[2] = n[1]$

$n[1] = n[0]$

for (int i = pos - 1; i > 0; i--){  
   $n[i] = n[i - 1]$ }



# Búsqueda lineal

- el array puede estar o no ordenado, ningún requisito inicial
- la búsqueda consiste en
  - se recorre uno a uno, desde el principio, los elementos del array
  - cada elemento se compara con el valor buscado
  - la búsqueda termina cuando se encuentra el elemento o cuando se acaba el array sin haberlo localizado

16	13	24	35	3	43	51	23	64	22	84	53	95	11	7
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Valor buscado = 3

# Búsqueda lineal

- el array puede estar o no ordenado, ningún requisito inicial
- la búsqueda consiste en
  - se recorre uno a uno, desde el principio, los elementos del array
  - cada elemento se compara con el valor buscado
  - la búsqueda termina cuando se encuentra el elemento o cuando se acaba el array sin haberlo localizado

16	13	24	35	3	43	51	23	64	22	84	53	95	11	7
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Valor buscado = 3

# Búsqueda lineal

- el array puede estar o no ordenado, ningún requisito inicial
- la búsqueda consiste en
  - se recorre uno a uno, desde el principio, los elementos del array
  - cada elemento se compara con el valor buscado
  - la búsqueda termina cuando se encuentra el elemento o cuando se acaba el array sin haberlo localizado

16	13	24	35	3	43	51	23	64	22	84	53	95	11	7
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Valor buscado = 3

# Búsqueda lineal

- el array puede estar o no ordenado, ningún requisito inicial
- la búsqueda consiste en
  - se recorre uno a uno, desde el principio, los elementos del array
  - cada elemento se compara con el valor buscado
  - la búsqueda termina cuando se encuentra el elemento o cuando se acaba el array sin haberlo localizado

16	13	24	35	3	43	51	23	64	22	84	53	95	11	7
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Valor buscado = 3

# Búsqueda lineal

- el array puede estar o no ordenado, ningún requisito inicial
- la búsqueda consiste en
  - se recorre uno a uno, desde el principio, los elementos del array
  - cada elemento se compara con el valor buscado
  - la búsqueda termina cuando se encuentra el elemento o cuando se acaba el array sin haberlo localizado

16	13	24	35	3	43	51	23	64	22	84	53	95	11	7
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Valor buscado = 3

# Búsqueda lineal

- el array puede estar o no ordenado, ningún requisito inicial
- la búsqueda consiste en
  - se recorre uno a uno, desde el principio, los elementos del array
  - cada elemento se compara con el valor buscado
  - la búsqueda termina cuando se encuentra el elemento o cuando se acaba el array sin haberlo localizado

16	13	24	35	3	43	51	23	64	22	84	53	95	11	7
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Valor buscado = 3

Posición = 4

# Búsqueda lineal

```
public boolean  buscarLineal(int[] números, int valorBuscado)
{
    boolean encontrado = false;
    int i = 0;
    while (i < números.length && ! encontrado) {
        if (números[i] == valorBuscado) {
            encontrado = true;
        }
        else {
            i++;
        }
    }
    return encontrado;
}
```



# Búsqueda lineal

```
public boolean  buscarLineal(int[] números, int valorBuscado)
{
    int i = 0;
    while (i < números.length) {
        if (números[i] == valorBuscado) {
            return true;
        }
        else {
            i++;
        }
    }
    return false;
}
```

## ■ Ejer 5.18

## Ejer 5.18

```
public int  buscarLineal(int[] números, int valorBuscado)
{
    int i = 0;
    while (i < números.length) {
        if (números[i] == valorBuscado) {
            return i;
        }
        else {
            i++;
        }
    }
    return -1;
}
```

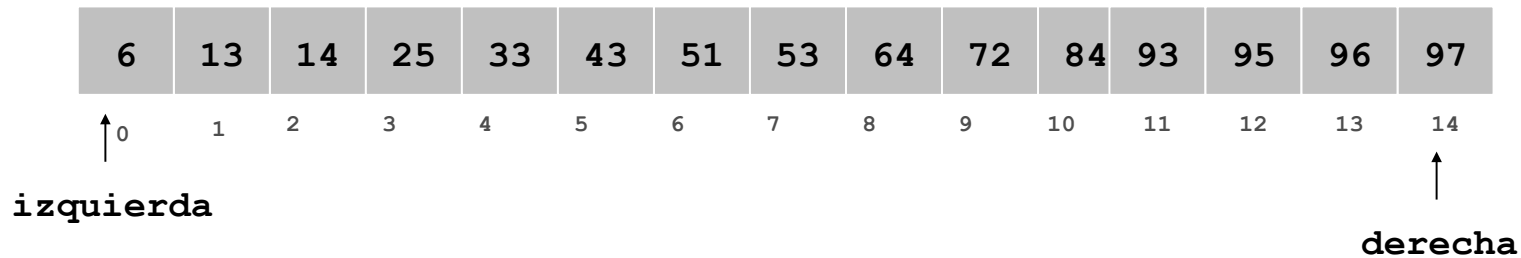
## Ejer 5.18

```
public int  buscarLineal(int[] números, int valorBuscado)
{
    for (int i = 0; i < numeros.length; i++) {
        if (numeros[i] == valorBuscado) {
            return i;
        }
    }
    return -1;
}
```

# Búsqueda binaria o dicotómica

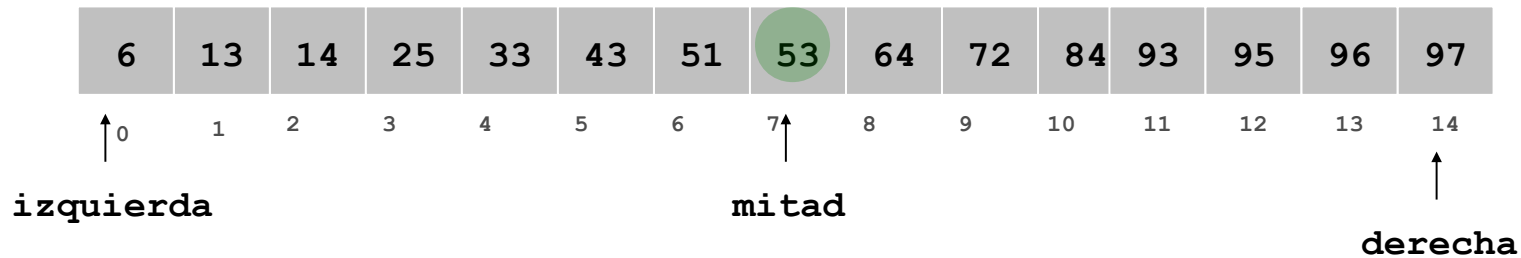
- Requisito: el array ha de estar **ordenado**
- Se compara el valor buscado con el que ocupa la posición central del array y:
  - si el valor coincide la búsqueda termina
  - si el valor buscado es menor que el de la posición mitad se reduce el intervalo de búsqueda a la izquierda de este valor
  - si el valor buscado es mayor que el de la posición mitad la búsqueda continúa por la derecha
  - este proceso se repite hasta **localizar el elemento** o hasta que el **intervalo de búsqueda quede anulado**.

# Búsqueda binaria o dicotómica



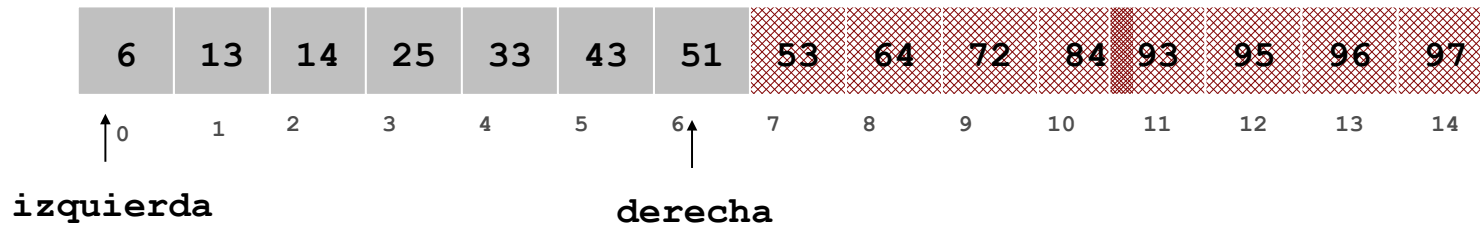
Valor buscado = 33

# Búsqueda binaria o dicotómica



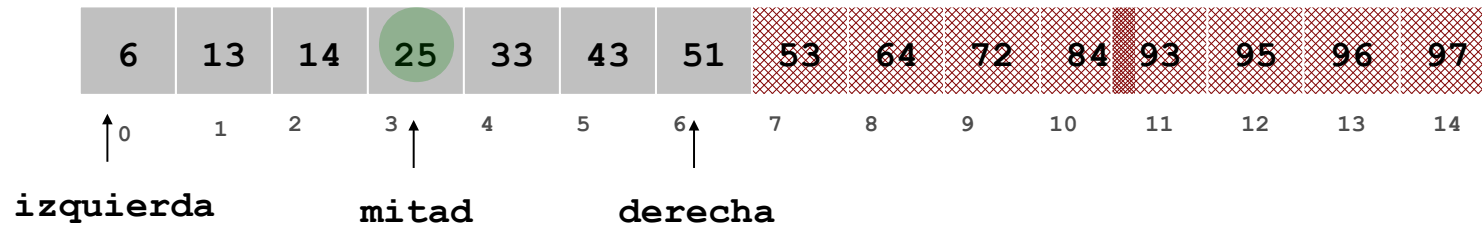
Valor buscado = 33

# Búsqueda binaria o dicotómica



Valor buscado = 33

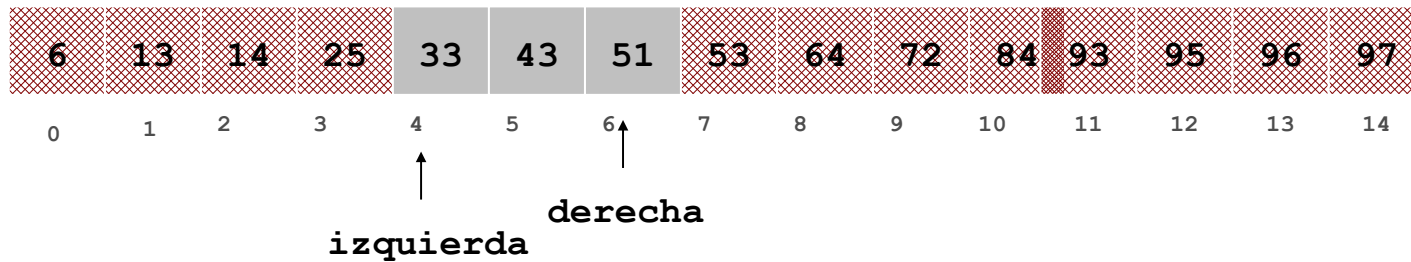
# Búsqueda binaria o dicotómica



Valor buscado = 33

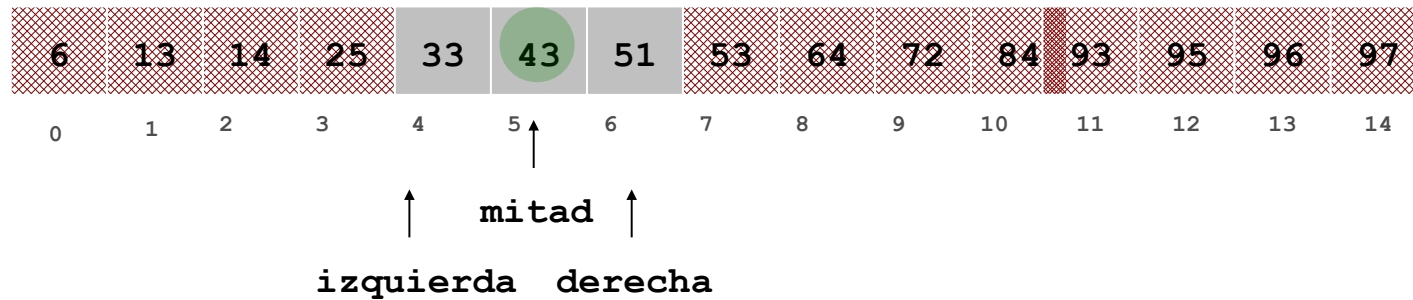


# Búsqueda binaria o dicotómica



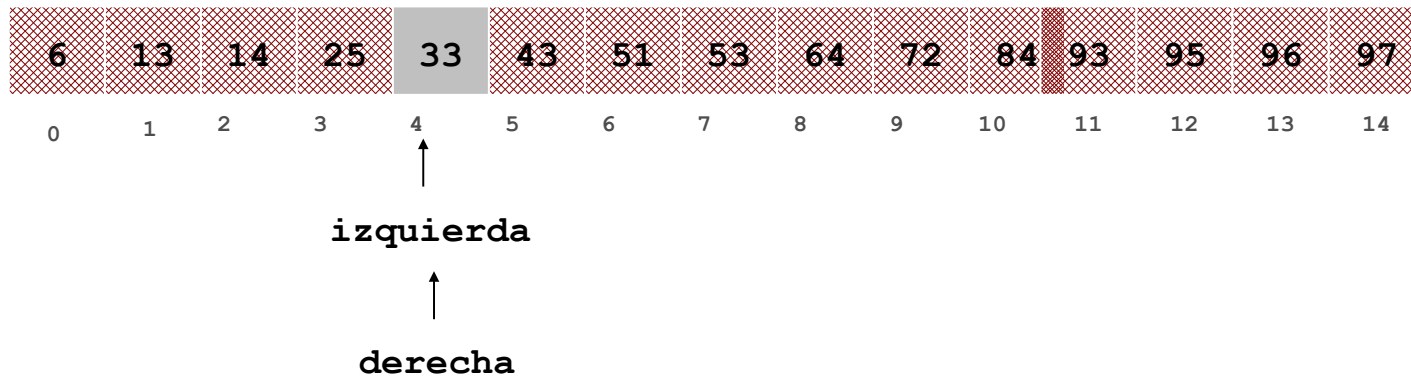
Valor buscado = 33

# Búsqueda binaria o dicotómica



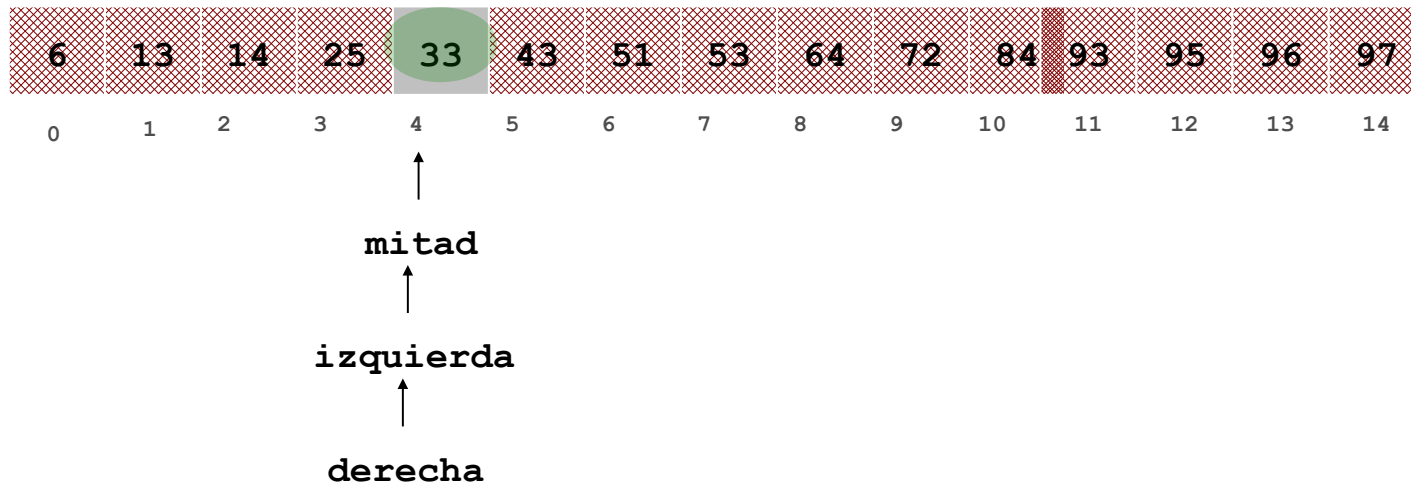
Valor buscado = 33

# Búsqueda binaria o dicotómica



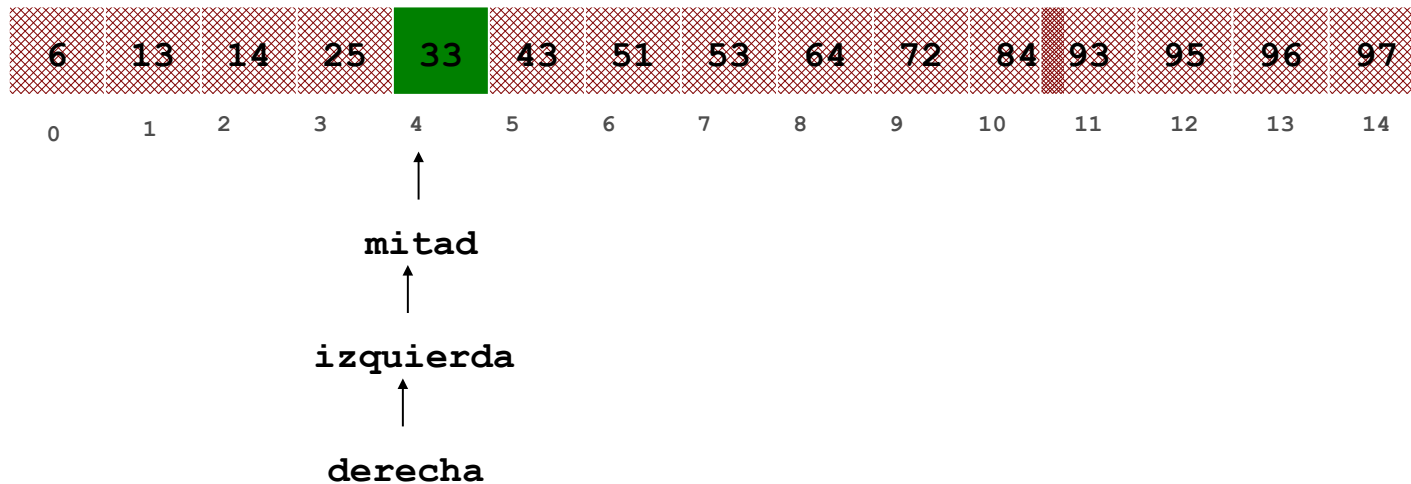
Valor buscado = 33

# Búsqueda binaria o dicotómica



Valor buscado = 33

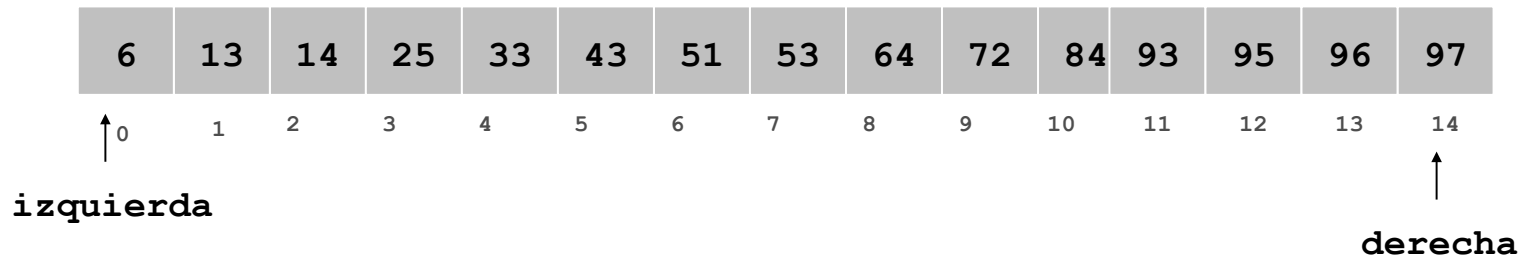
# Búsqueda binaria o dicotómica



Valor buscado = 33

Posición = 4

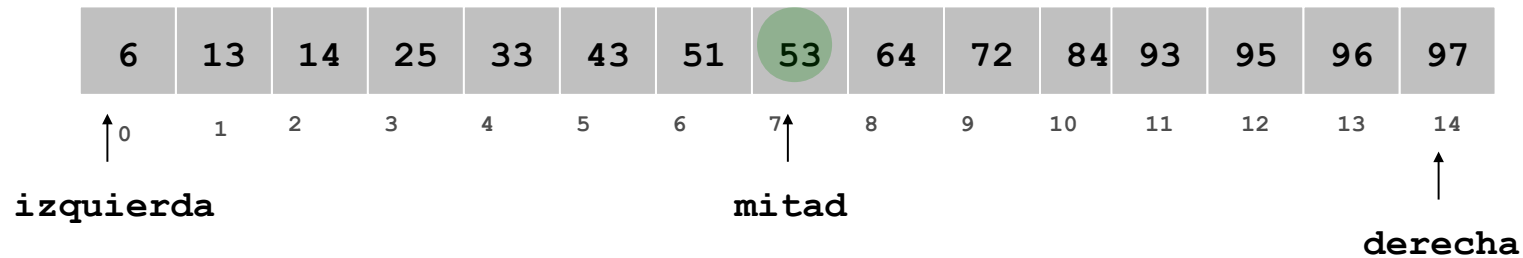
# Búsqueda binaria o dicotómica



Valor buscado = 32

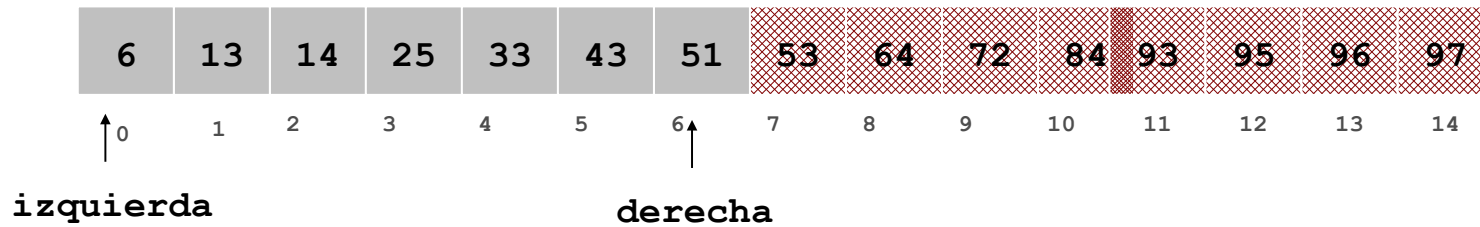
**Buscamos ahora  
un valor que no está  
en el array**

# Búsqueda binaria o dicotómica



Valor buscado = 32

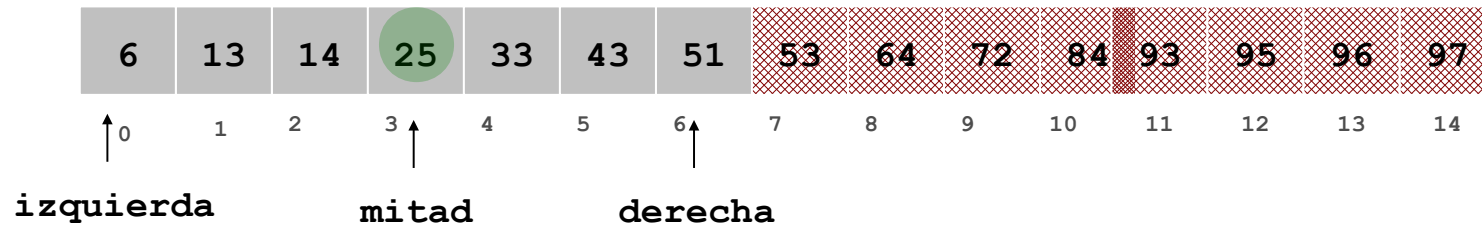
# Búsqueda binaria o dicotómica



Valor buscado = 32

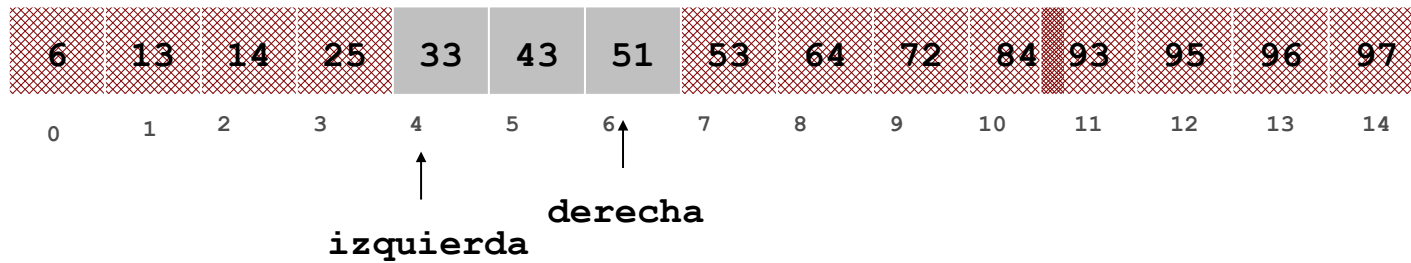


# Búsqueda binaria o dicotómica



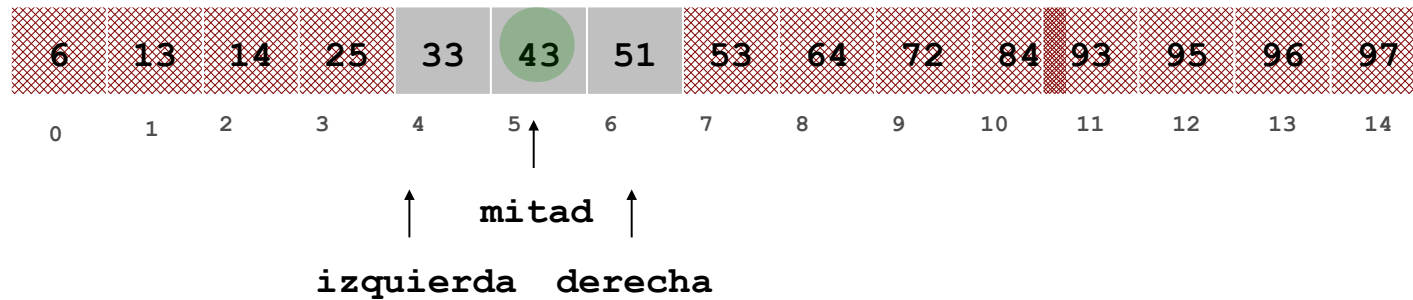
Valor buscado = 32

# Búsqueda binaria o dicotómica



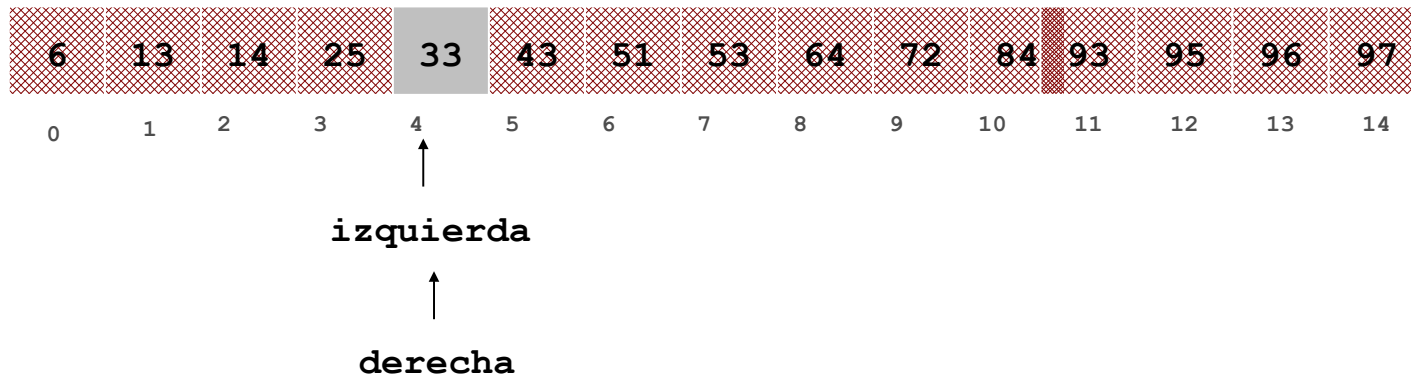
Valor buscado = 32

# Búsqueda binaria o dicotómica



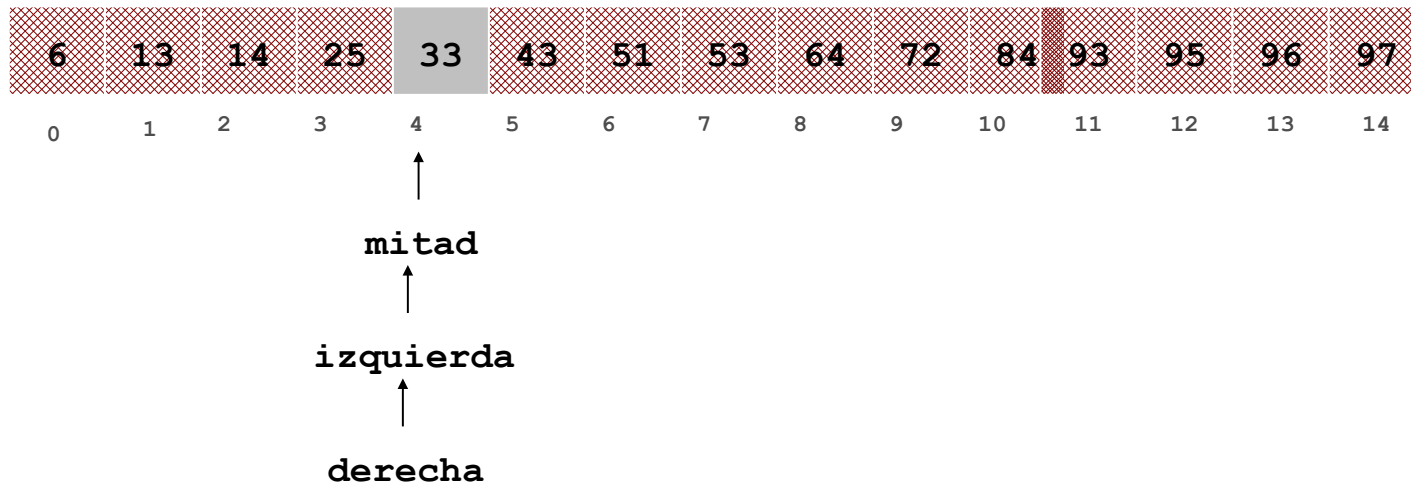
Valor buscado = 32

# Búsqueda binaria o dicotómica



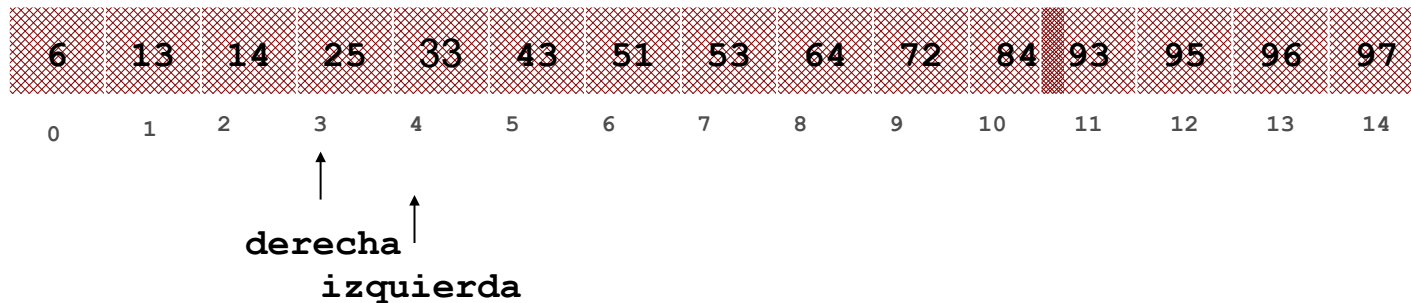
Valor buscado = 32

# Búsqueda binaria o dicotómica



Valor buscado = 32

# Búsqueda binaria o dicotómica



Valor buscado = 32

**Ahora izquierda > derecha**

**Ya no hay intervalo de  
búsqueda**

**El elemento buscado no existe**

# Búsqueda binaria o dicotómica

```
public boolean  buscarDicotomica(int[] números, int valorBuscado)
{
    boolean encontrado = false;
    int izquierda = 0;
    int derecha = numeros.length - 1;
    while  (izquierda <= derecha  && !encontrado)    {
        int mitad = (izquierda + derecha) / 2;
        if (numeros[mitad] == valorBuscado)  {
            encontrado = true;
        }
        else if (numeros[mitad] > valorBuscado) {
            derecha = mitad - 1;
        }
        else  {
            izquierda = mitad + 1;
        }
    }
    return encontrado;
}
```

# Búsqueda binaria o dicotómica

```
public boolean  buscarDicotomica(int[] números, int valorBuscado)
{
    int izquierda = 0;
    int derecha = numeros.length - 1;
    while (izquierda <= derecha) {
        int mitad = (izquierda + derecha) / 2;
        if (numeros[mitad] == valorBuscado)  {
            return true;
        }
        else if (numeros[mitad] > valorBuscado)    {
            derecha = mitad - 1;
        }
        else {
            izquierda = mitad + 1;
        }
    }
    return false;
}
```

■ Ejer 5.19



# Búsqueda binaria o dicotómica

- Java proporciona en la clase `Arrays` (consultar API)
  - **`Arrays.binarySearch()`**
  - ```
int[] numeros = {-3, 2, 8, 12, 17, 29, 44, 58, 79};  
int index = Arrays.binarySearch(numeros, 29);  
System.out.println("29 se ha encontrado en posición " + index);
```

Si no existe devuelve:  
`-(1 + array.length)`

- Ejer 5.19

# Insertar un valor en una posición

- `public void insertarEnPosicion(int valor, int p)`
  - el array no debe estar completo
  - posición **p** debe ser correcta

|   |    |    |    |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|----|----|----|
| 6 | 13 | 14 | 25 | 33 | 43 | 51 | 53 | 72 | 84 |    |    |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 |

pos = 10

insertar 27 en posición p = 4

# Insertar un valor en una posición

- `public void insertarEnPosicion(int valor, int p)`
  - el array no debe estar completo
  - posición **p** debe ser correcta

|   |    |    |    |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|----|----|----|
| 6 | 13 | 14 | 25 | 27 | 33 | 43 | 51 | 53 | 72 | 84 |    |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 |

pos = 11

elemento ya  
insertado

**Hacemos hueco al 27 desplazando  
todos a la derecha desde la posición p  
hasta el final**

# Insertar un valor en una posición

```
public void insertarEnPosición(int valor, int p)
{
    if (!estaLlena()) {
        if (p >= 0 && p <= pos) { // posición correcta
            for (int i = pos - 1; i >= p; i--) {
                numeros[i + 1] = numeros[i];
            }
            numeros[p] = valor;
            pos++;
        }
    }
}
```

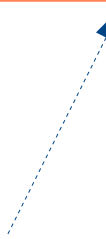
# Insertar un valor en una posición (usando `System.arraycopy()` )

```
public void insertarEnPosición(int valor, int p)
{
    if (!estaLlena()) {
        if (p >= 0 && p <= pos) { // posición correcta

            System.arraycopy(numeros, p, numeros, p + 1, pos - p);

            numeros[p] = valor;
            pos++;
        }
    }
}
```

el mismo efecto que desplazar a la derecha



# Insertar un valor en orden

- `public void insertarEnOrden(int valor)`
  - el array no debe estar completo

|   |    |    |    |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|----|----|----|
| 6 | 13 | 14 | 25 | 33 | 43 | 51 | 53 | 72 | 84 |    |    |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 |

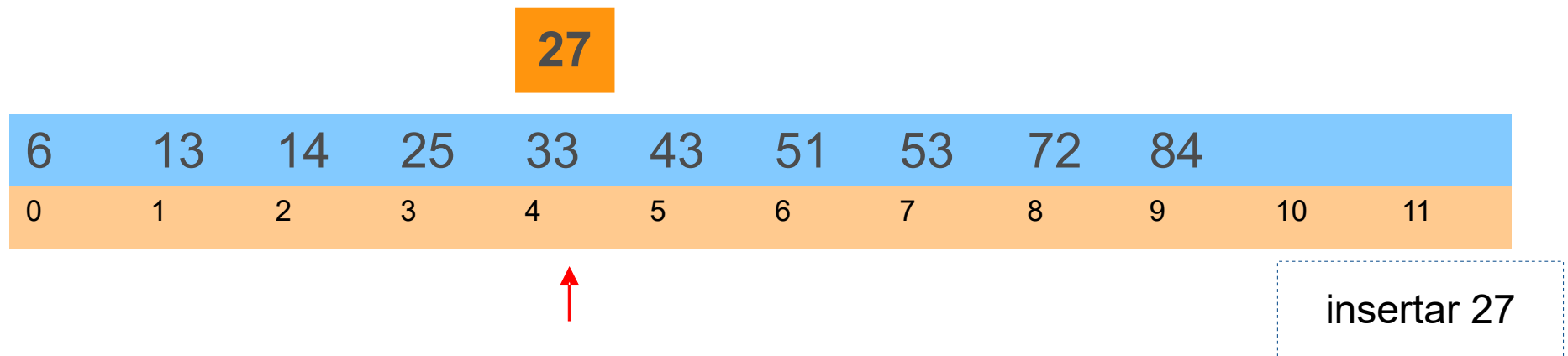
insertar 27

|   |    |    |    |           |    |    |    |    |    |    |    |
|---|----|----|----|-----------|----|----|----|----|----|----|----|
| 6 | 13 | 14 | 25 | <b>27</b> | 33 | 43 | 51 | 53 | 72 | 84 |    |
| 0 | 1  | 2  | 3  | 4         | 5  | 6  | 7  | 8  | 9  | 10 | 11 |

elemento ya  
insertado

# Insertar un valor en orden –

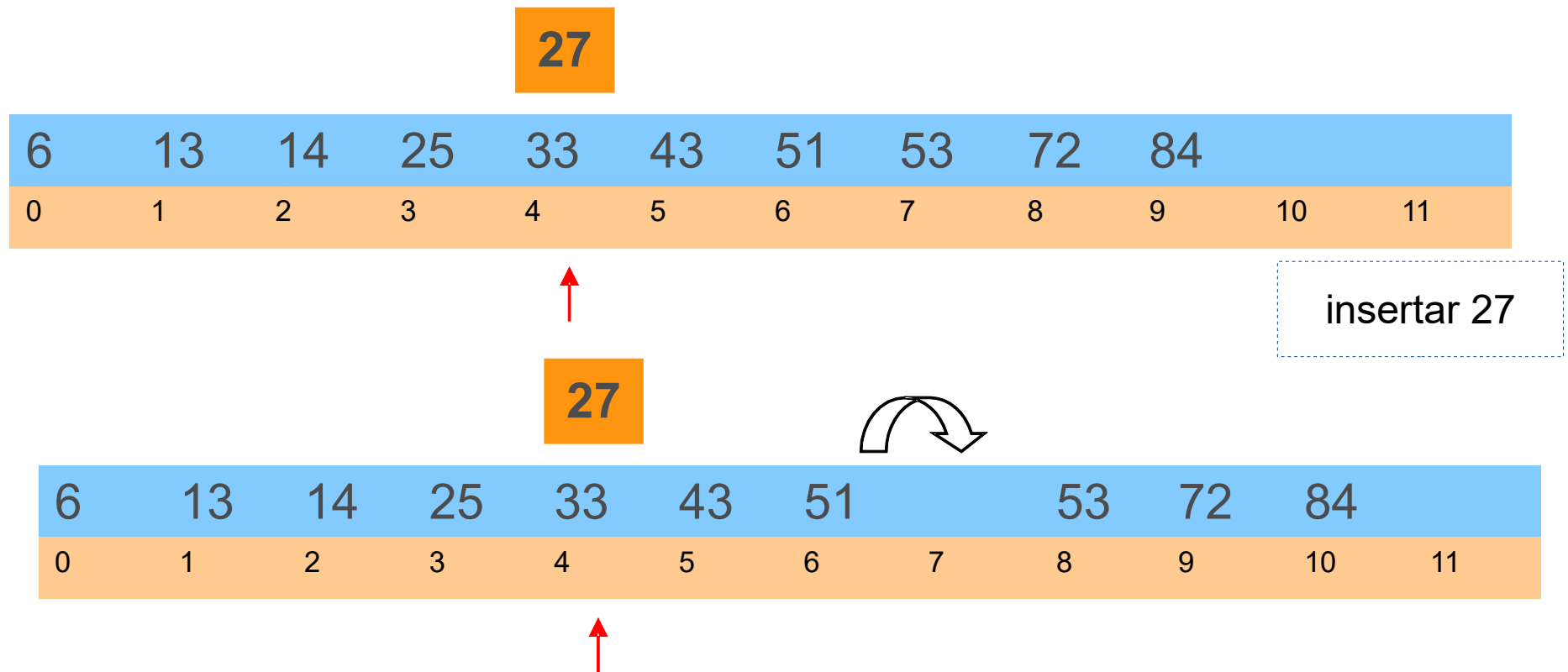
## a) buscar lugar adecuado para insertar



# Insertar un valor en orden

## b) desplazar elementos hacia la derecha

- desplazamos hacia la derecha empezando desde el último




- Los pasos a) y b) se pueden combinar
  - se busca hueco para insertar mientras se desplaza



# Insertar un valor en orden

## c) insertar el valor en su lugar

|   |    |    |    |           |    |    |    |    |    |    |    |
|---|----|----|----|-----------|----|----|----|----|----|----|----|
| 6 | 13 | 14 | 25 | <b>27</b> | 33 | 43 | 51 | 53 | 72 | 84 |    |
| 0 | 1  | 2  | 3  | 4         | 5  | 6  | 7  | 8  | 9  | 10 | 11 |



# Insertar un valor en orden

```
public void insertarEnOrden(int valor)
{
    if (!estaCompleto() && !estaElemento(valor)) {
        // buscar hueco mientras se desplaza
        int i = pos - 1;
        while (i >= 0 && elementos[i] > valor) {
            elementos[i + 1] = elementos[i];
            i--;
        }
        elementos[i + 1] = valor; // insertar el valor
        pos++; // incrementamos el nº de elementos en la lista
    }
}
```

|   |    |    |    |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|----|----|----|
| 6 | 13 | 14 | 25 | 33 | 43 | 51 | 53 | 72 | 84 |    |    |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 |

# Ordenación de arrays

---

- Ordenar un array
  - reagrupar sus elementos en un determinado orden
    - ascendente
    - descendente
- Múltiples algoritmos de ordenación
  - ordenación por **selección directa** (o selección del mínimo)
  - ordenación por **inserción directa**

# Ordenación por selección directa (o selección del mínimo)

- Varios pasos de ordenación
- Se **selecciona el menor** del array
- Se **intercambia** con el valor de la primera posición
- Se selecciona el siguiente menor del array (de entre los que quedan por ordenar)
- Se intercambia con el valor de la segunda posición
- Se repite el proceso anterior hasta que todos los valores están colocados

# Ordenación por selección directa (o selección del mínimo)

24

12

36

5

7

15

Paso 1

# Ordenación por selección directa (o selección del mínimo)

|    |    |    |    |   |    |        |
|----|----|----|----|---|----|--------|
| 24 | 12 | 36 | 5  | 7 | 15 | Paso 1 |
| 5  | 12 | 36 | 24 | 7 | 15 | Paso 2 |

# Ordenación por selección directa (o selección del mínimo)

|    |    |    |    |    |    |        |
|----|----|----|----|----|----|--------|
| 24 | 12 | 36 | 5  | 7  | 15 | Paso 1 |
| 5  | 12 | 36 | 24 | 7  | 15 | Paso 2 |
| 5  | 7  | 36 | 24 | 12 | 15 | Paso 3 |

# Ordenación por selección directa (o selección del mínimo)

|       |       |       |       |       |       |        |
|-------|-------|-------|-------|-------|-------|--------|
| 24    | 12    | 36    | 5     | 7     | 15    | Paso 1 |
| 5     | 12    | 36    | 24    | 7     | 15    | Paso 2 |
| 5     | 7     | 36    | 24    | 12    | 15    | Paso 3 |
| 5     | 7     | 12    | 24    | 36    | 15    | Paso 4 |
| ..... | ..... | ..... | ..... | ..... | ..... |        |

esta paso  
no cuenta



# Ordenación por selección directa (o selección del mínimo)

```
/**
 * Ordenar en orden ascendente
 */
public void ordenarSeleccionDirecta(int[] array)
{
    for (int i = 0; i < array.length - 1; i++) {
        int posmin = i;
        for (int j = i + 1; j < array.length; j++) {
            if (array[j] < array[posmin]) {
                posmin = j;
            }
        }
        int aux = array[posmin];
        array[posmin] = array[i];
        array[i] = aux;
    }
}
```

for marca nº pasos de ordenación

5 pasos

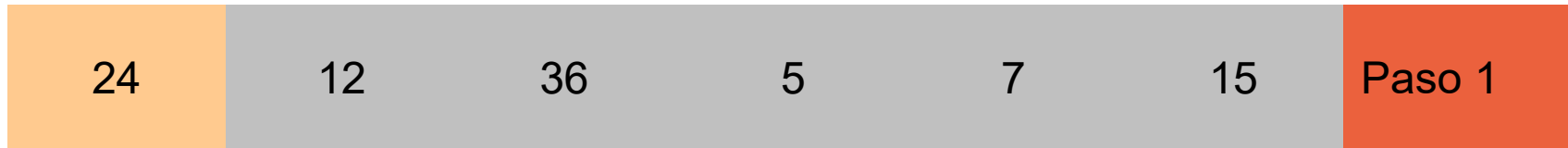
Desde aquí  
se empieza  
a verificar  
los restantes

selección del  
mínimo

intercambiar el  
mínimo  
(que está en  
posición *posmin*)  
por el elemento de  
posición *i*

# Ordenación por inserción directa

---



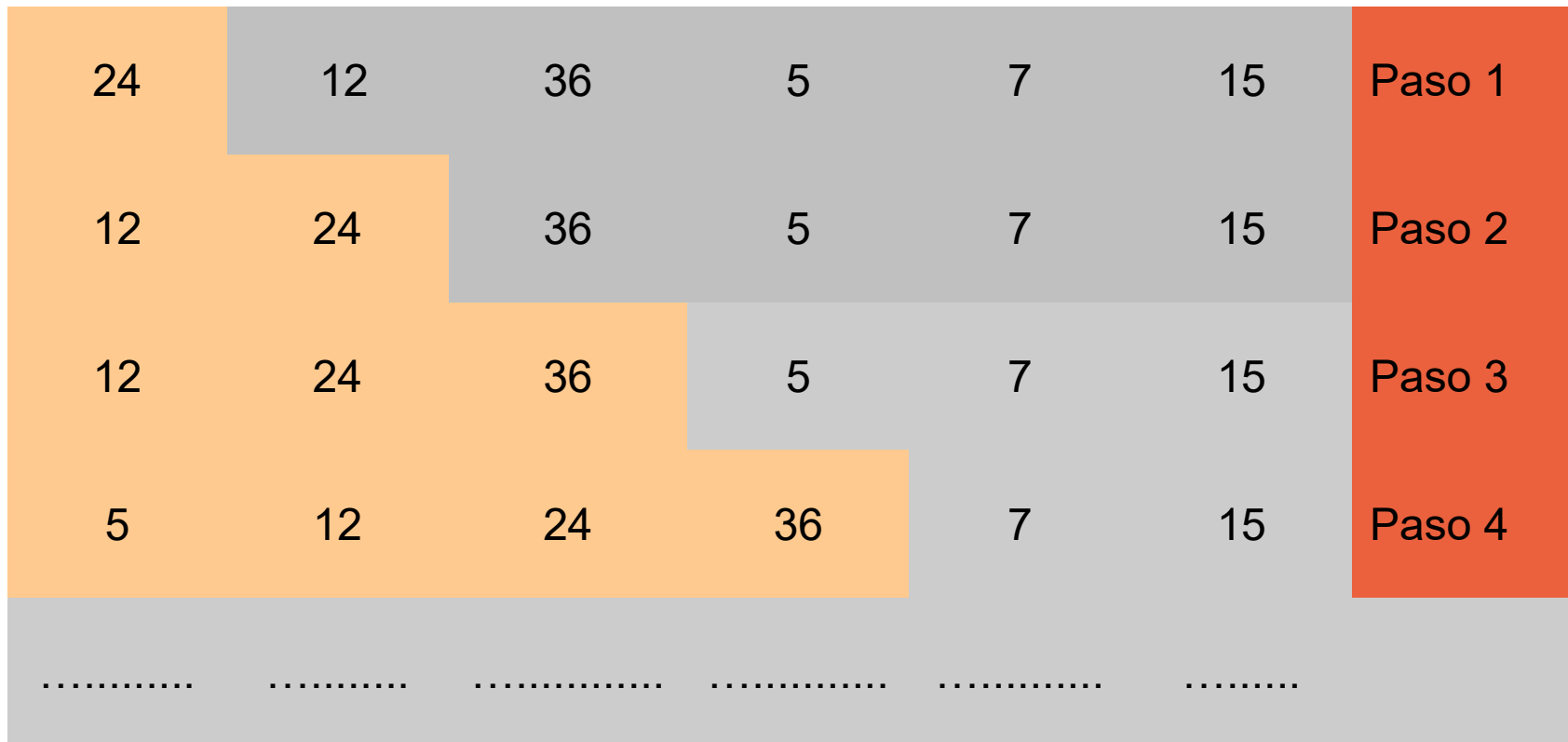
# Ordenación por inserción directa

|    |    |    |   |   |    |        |
|----|----|----|---|---|----|--------|
| 24 | 12 | 36 | 5 | 7 | 15 | Paso 1 |
| 12 | 24 | 36 | 5 | 7 | 15 | Paso 2 |

# Ordenación por inserción directa

|    |    |    |   |   |    |        |
|----|----|----|---|---|----|--------|
| 24 | 12 | 36 | 5 | 7 | 15 | Paso 1 |
| 12 | 24 | 36 | 5 | 7 | 15 | Paso 2 |
| 12 | 24 | 36 | 5 | 7 | 15 | Paso 3 |

# Ordenación por inserción directa



# Ordenación por inserción directa

```
/**
 * Ordenar en orden ascendente
 */
public void ordenarInsercionDirecta(int[] array)
{
    for (int i = 1; i < array.length; i++) {
        int aux = array[i];
        int j = i - 1;
        while (j >= 0 && array[j] > aux) {
            array[j + 1] = array[j];
            j--;
        }
        array[j + 1] = aux;
    }
}
```

for marca nº pasos de ordenación

Buscar hueco  
para *aux* a la  
vez que se  
desplaza a la derecha

## ■ Ejer 5.20

# Ejercicios adicionales

---

EJAD04 CD Canción

EJAD05 TestDosDimensiones