INICIACIÓN A LA PROGRAMACIÓN
LENGUAJE JAVA con BlueJ

Tema 3 *Clases y Objetos*

Tema 4 *Comunicación entre objetos. Algoritmos*

Tema 5 *Herencia y abstracción de datos*

Tema 6 *Diseño de clases*

TEMA 4 : Comunicación y algoritmos

1. Introducción a la programación estructurada
   - Estructuras de repetición en JAVA
   - Entrada de datos en JAVA
2. Comunicación entre objetos
   - Visibilidad. Métodos **`static`**
   - Referencia **`this`**
   - Método **`main`**
   - Modificadores de visibilidad
3. Arrays
   - Unidimensionales
   - Multidimensionales
4. Algoritmos en Arrays
   - Búsquedas
   - Ordenación
   - Análisis de secuencias

TEMA 4 : Comunicación y algoritmos

Introducción a la programación estructurada

La **programación imperativa** (conjunto de instrucciones que se ejecutan de forma secuencial) es la forma de programar más antigua y la que mejor refleja la arquitectura de los microprocesadores, pero:

- *En cuanto aumenta el tamaño de los proyectos se hace difícil su gestión y mantenimiento*

La **programación estructurada** nace con el objetivo de:

- Buscar una forma de estructurar mejor los programas
- Que sean más fáciles de entender por uno mismo y por los demás.

Prog. Estructurada → Modular (basada en objetos) → Prog. Orientada a Objetos

TEMA 4 : Comunicación y algoritmos

Introducción a la programación estructurada

Para satisfacer sus objetivos, la **programación estructurada:**

- Sólo considera *tres tipos de estructuras de control:*
  Secuencia, Selección y Repetición.
- *Con un solo punto de entrada y un solo punto de salida.*

Java tiene 7 estructuras de control:

- Estructura de secuencia: instrucciones ejecutadas una tras otra.
- Estructuras de selección: Se salta de una parte a otra del código si se cumple una cierta condición : **if ,if/else, switch (vistas en tema 3)**
- Estructuras de repetición: Se repite la ejecución de una instrucción si se cumple una cierta condición: **while, do/while, for**

TEMA 4 : Comunicación y algoritmos

Estructuras de repetición en JAVA (I)

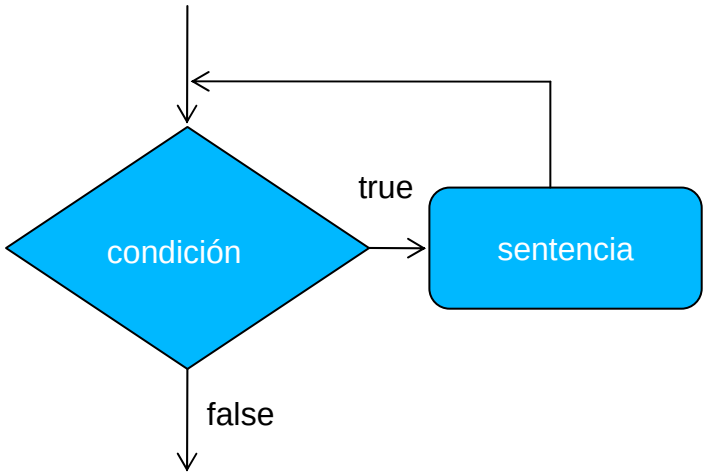
WHILE:

Permite al programador especificar que una acción se repita en tanto se cumpla una condición.

```
while (condicion) sentencia;
while (condicion){
   sentencia_1;
   ……
   sentencia_n;
}
```

Ejemplo: Encontrar la primera potencia de 2 mayor que 1000.

```
int producto = 2;
while (producto <= 1000)
    producto = 2 * producto;
```



TEMA 4 : Comunicación y algoritmos

Estructuras de repetición en JAVA (II)

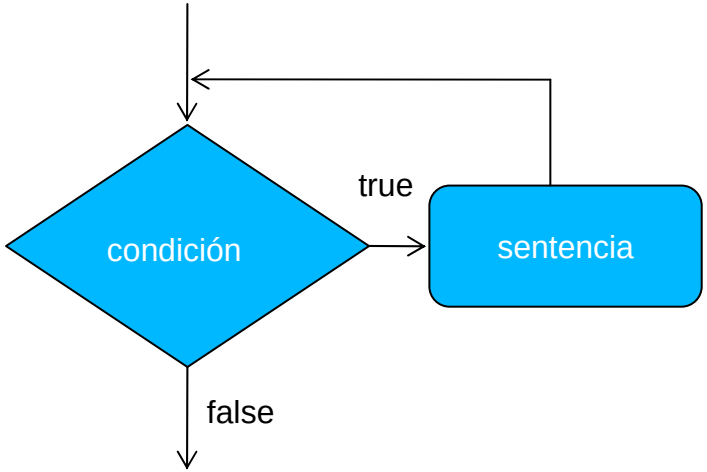
DO / WHILE:

Permite al programador especificar que una acción se repita en tanto se cumpla una condición.

```
do
   sentencia;
while (condicion);
do{
   sentencia_1;
   ……
   sentencia_n;
} while (condicion)
```

Ejemplo: Encontrar la primera potencia de 2 mayor que 1000.

```
int producto = 2;
do
    producto = 2 * producto;
while (producto <= 1000)
```



Diferencia con sentencia **while:** La *sentencia o sentencias se ejecutan al menos una vez. La condición se comprueba después* de haber realizado la primera iteración.

TEMA 4 : Comunicación y algoritmos

Estructuras de repetición en JAVA (III)

FOR:

Permite una repetición controlada por un contador:

```
for (inicialización contador;
  condición;
  expresión incremento contador)
     sentencia;
for (inic. cont.; cond.; inc. cont.){
 sentencia_1;
 ……
 sentencia_n;
}
```

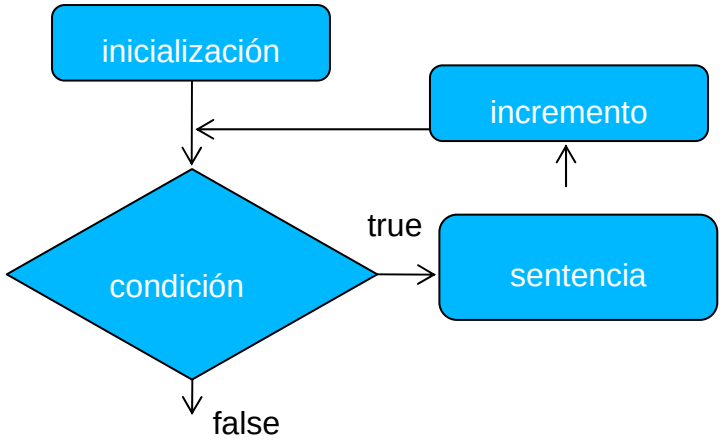
Ejemplos:

Variable de control de 7 a 77 en incrementos de 7.

```
for (int i = 7; i <= 77; i += 7)
{…};
```

Variable de control de 99 a 0 en incrementos de -11

```
for (int i = 99; i >= 0; i = i - 11)
{…};
```



TEMA 4 : Comunicación y algoritmos

Estructuras de repetición en JAVA (IV)

```
for (int contador = 1; contador <= 10; contador ++)
   sentencia;
```

es equivalente a:

```
int contador = 1;
while (contador <= 10) {
   sentencia;
   contador++;
}
```

La elección de una estructura de repetición:

- Si número fijo de veces, elegir **`for`**
- Si 1 o más veces, elegir **`do { } while { }`**
- Si 0 o más veces, elegir **`while ( ) { }`**

Se elegirá la que mejor se ajuste al concepto para mejorar la legibilidad

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA

- **Leer de datos entrada estándar:** Para leer datos de la entrada estándar (teclado) se usa:
  - Objeto ***System.in***
  - Clase ***Scanner***
- Se crea un objeto de la clase *Scanner* mediante el objeto *System.in*
- Se usan los métodos de la clase Scanner para leer datos.
- La clase *Scanner* pertenece al paquete *java.util*

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA: Ejemplo (1)

```
import java.util.*; //necesario para Scanner
  . . . . . .

  Scanner scan = new Scanner(System.in);
  //leer un entero
  int n = scan.nextInt();
  //leer un String
  String next = scan.next();
```

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA: Clase **scanner** (I)

- La clase *Scanner* se usa para *escanear textos*.
- Con un scanner se divide el texto en “trozos” (o **tokens)** según unos delimitadores.
- Por defecto los delimitadores son: espacios en blanco, tabuladores, saltos de línea.
- Los delimitadores se descartan cuando se escanea el texto.
- Ejemplo: si tenemos el texto

pepe 234.67 hola -34 adiós <INTRO>

Los tokens serían: pepe, 234.67, hola, -34, adiós

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA: Clase **scanner** (II)

- **Métodos de Scanner:**
  - `String next()`: retorna un String con el siguiente token.
  - `int nextInt()`: retorna el siguiente token transformándolo a `int`.
  - `double nextDouble()`: retorna el siguiente token transformándolo a `double`.
  - `...`

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA: Ejemplo (2)

Teclado

14 -28.2 hola adiós <INTRO>

```
Scanner scan = new Scanner(System.in);
int n = scan.nextInt(); // n es igual a 14
double real = scan.nextDouble();//real igual a -
  28.2
String next = scan.next(); //next igual a “hola”
String s = scan.next(); //s igual a “adiós”
```

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA: otros modos

- Otra forma de pedir datos mediante el teclado es usando adecuadamente la clases
  - **JOptionPane**
  - Scanner

- Mediante JOptionPane construimos ventanas donde introducir datos.

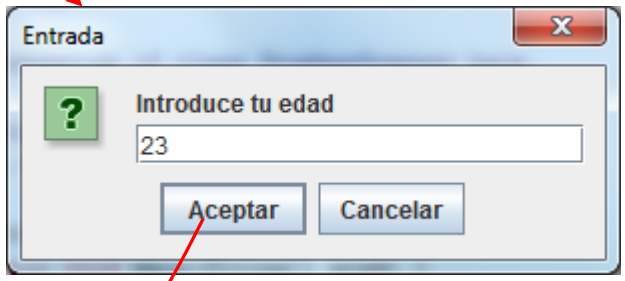
- JOptionPane pertence al paquete javax.swing

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA: Ejemplo (3)

```
import javax.swing.*; //necesario para JOptionPane
import java.util.*;//necesario para Scanner
   . . . . . .
String datos = JOptionPane.showInputDialog("Introduce tu edad");
Scanner scan = new Scanner(datos);
int edad = scan.nextInt();
```

Aparece ventana



Al pulsar "Aceptar" continua

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA: Excepciones (I)

- ¿Qué ocurre si intentamos leer un número entero del teclado y el usuario introduce ‘Pepe’.
- El método *nextInt()* no puede transformar ‘Pepe’ en un entero.

- Se produce un **error en tiempo de ejecución** abortando el programa (una excepción).

- En lenguaje Java se dice que: *el método nextInt() ha lanzado una excepción*.

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA: Excepciones (II)

- Las excepciones se lanzan y si no se ***capturan*** el programa aborta.
- Se capturan mediante bloque

```
try{…} catch(Nombre_Excepcion e){…}
```

- Dentro del bloque *try* están las instrucciones que pueden lanzar la excepción.
- Dentro del bloque *catch* las acciones a realizar en caso de que se lance.

  - Veremos las excepciones con más detenimiento en temas posteriores. Ahora, un ejemplo sencillo

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA: Ejemplo (4)

```
Scanner scan = new Scanner(System.in);
int edad = 0;
try {
    System.out.print(“Introduce tu edad: “);
    edad = scan.nextInt();
    System.out.println(“Tu edad es “ + edad);
} catch(Exception e) {
    System.out.println(“Error: edad mal introducida.”);
}
. . . .//continua el programa
```

Con excepciones

No se ejecuta nada entre el punto que se lanza excepción y catch

Sin excepciones

TEMA 4 : Comunicación y algoritmos

Entrada de datos en JAVA: Ejemplo (5)

Ejemplo de entrada robusta:

```
int edad = 0;
boolean correcto = false;
while (!correcto) {
  try {
      Scanner scan = new Scanner(System.in);
      System.out.print(“Introduce edad:”);
      edad = scan.nextInt();
      correcto = true;
  } catch (Exception e) {
      correcto = false;
  }
}
System.out.println(“Tu edad es “ + edad + “ años”);
```

TEMA 4 : Comunicación y algoritmos

Comunicación entre objetos (I)

- Recordamos que sabemos sobre los objetos (tema 3) que…

Un objeto es una entidad que :
1)tiene una estructura (campos)
2)con unos valores (estado)
3)un comportamiento (métodos).

1. El estado viene dado por los valores que tomen en un momento dado los distintos atributos del objeto.
2. El comportamiento de un objeto viene marcado por la secuencia de acciones y reacciones que tienen lugar a lo largo del ciclo de vida del objeto, que dependen de su estado y tienen efecto sobre él.

Pero ¿Cómo se actúa sobre dicho comportamiento?

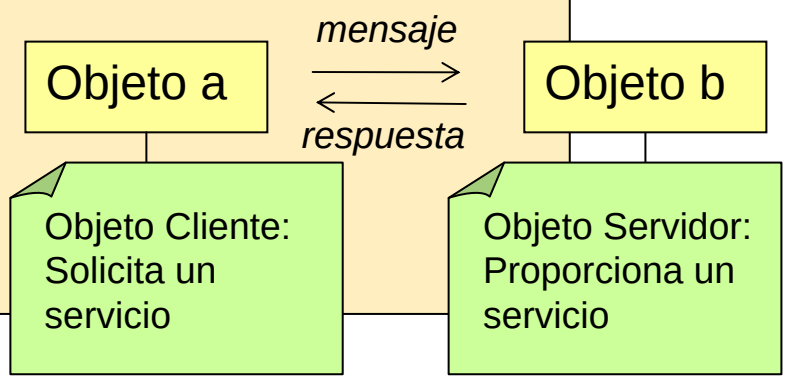
Comunicación entre objetos (II)

- Recordamos que sabemos sobre los métodos (tema 3) que…
  1. El comportamiento se expresa como un ***conjunto de métodos*** *definidos en la* clase del objeto.
  2. Donde los métodos son una sección de código autocontenida que puede recibir datos de entrada y producir datos de salida → ***Abstracción funcional***

¿Cómo puede un objeto acceder a las operaciones de otro objeto?

→ ***Se comunican por medio de mensajes.***

1. Si un objeto desea invocar los métodos de otro objeto debe enviarle un mensaje.
2. Cada mensaje enviado a un objeto debe corresponderse con un método definido en la interfaz del objeto receptor.



Comunicación entre objetos (III)

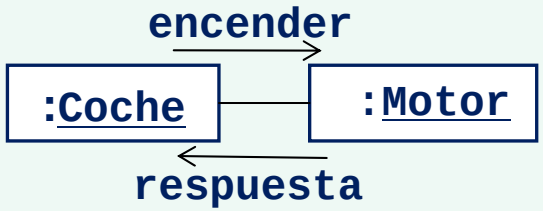
```
class Coche {
 Motor motor = new Motor();
 boolean encendido = false;

 public void arrancar(){
  encendido =
  motor.encender();
 }
}


class Motor {
 public boolean encender(){
   return true;
 }
}
```

Los programas en Java modelan el mundo real como objetos que interaccionan enviándose mensajes → programación basada en objetos.

- Si un objeto desea invocar los métodos de otro objeto debe enviarle un mensaje.
- Cada mensaje enviado a un objeto debe corresponderse con un método definido en la interfaz del objeto receptor.

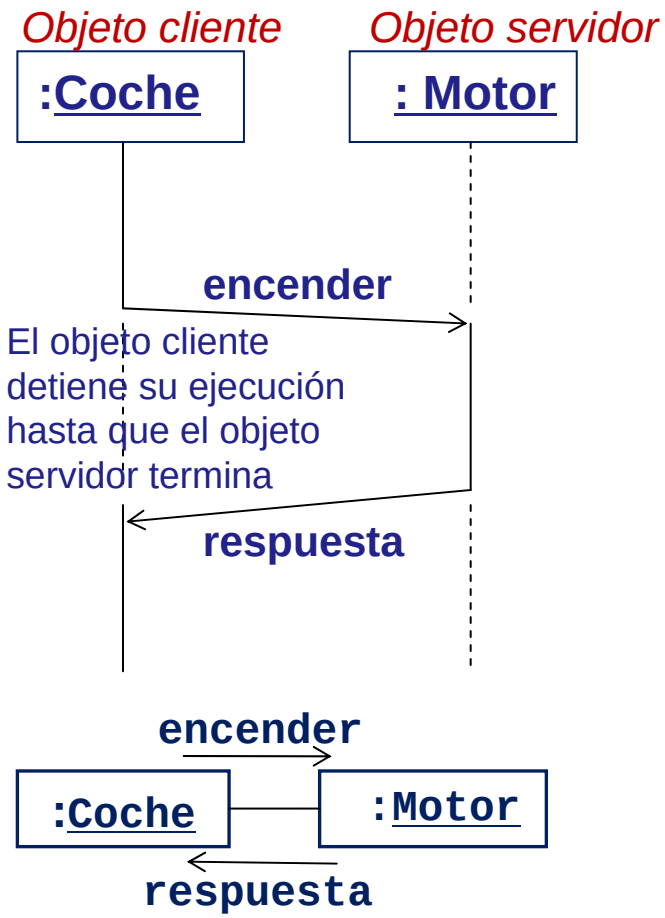


Comunicación entre objetos (IV)

objetoServidor.metodo(argumentos)

¿Qué ocurre en la llamada?

- El objeto cliente queda a la espera de la finalización del método.
- Cuando el servidor termina el objeto cliente continúa su ejecución.
- El objeto cliente sólo conoce la interfaz del método y la funcionalidad asociada (*qué hace)* pero no debe saber nada acerca del *cómo se hace.*



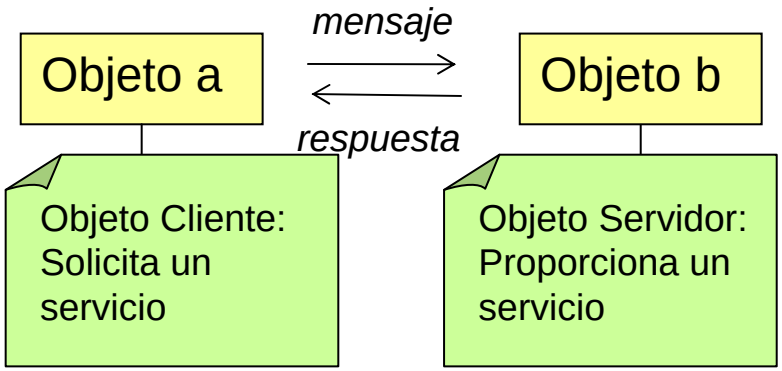
Comunicación entre objetos (V)

- EJEMPLO: Obtener las iniciales del nombre de un cliente

Equivale a enviar un mensaje a un objeto Persona, que calcule sus iniciales y nos las devuelva para poder procesarlas según convenga.

Esto es:

```
String iniciales;
iniciales = unaPersona.obtenerIniciales();
```



TEMA 4 : Comunicación y algoritmos
Visibilidad. Métodos **`static`** (I)

→ En lo visto hasta ahora, la invocación de un método implica la existencia previa de un objeto.

→ Si la invocación de un método es un mensaje, es necesario que exista un objeto receptor para el mismo. Sin embargo, esto es

- Excesivo en algunas circunstancias.
- Inconveniente en otras.

Excesivo:

→ Supongamos que deseamos acceder a funciones matemáticas:

→ No es posible hacer simplemente **`y = cos(x),`** sino **`y = a.cos(x),`** donde **`a`** es el objeto que puede calcular el coseno

TEMA 4 : Comunicación y algoritmos
Visibilidad. Métodos **`static`** (II)

Inconveniente:

- Supongamos que tenemos una aplicación que es un juego de guerra.
- El juego modela la realidad de tal manera que los soldados (*objetos de la* **`clase Soldado`**) se comportan de una manera u otra dependiendo del número de soldados que *"quedan vivos".*

→ Es necesario llevar la cuenta de los **`soldados:`**

- Variable **`int efectivos`**

→ Es necesario mantener de forma consistente el valor de la variable **`efectivos`** en cada uno de los **`soldados`** de la aplicación, lo cual no es trivial.

TEMA 4 : Comunicación y algoritmos
Visibilidad. Métodos **`static`** (III)

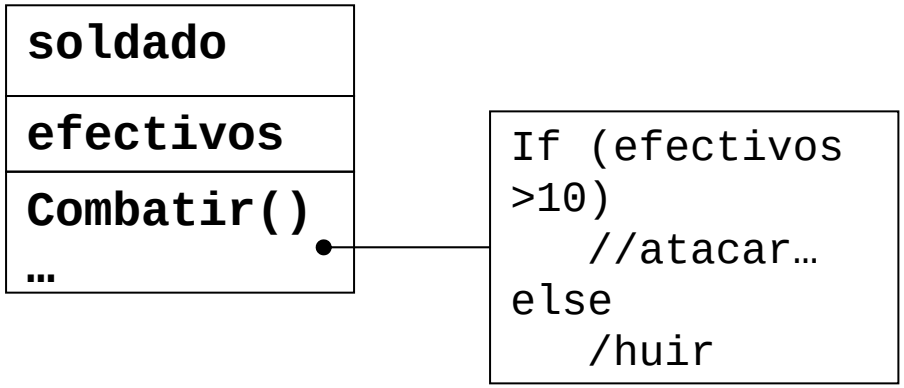
Supongamos que tenemos 100 soldados.

Cualquier baja o incorporación supone actualizar 100 variables, tantas como objetos soldado.

Varias alternativas, todas malas:

1.Cada vez que un soldado causa baja o se incorpora a la unidad se lo notifica a todos los demás.

2.Existe un objeto supervisor o controlador que realiza tales notificaciones.
3....

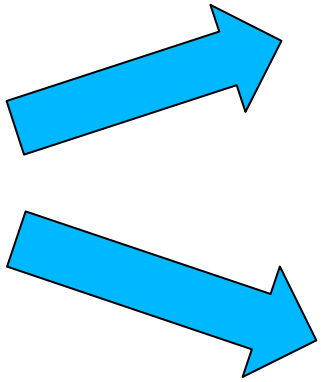


¿Cómo podemos hacer?

Mediante campos y métodos `static`

TEMA 4 : Comunicación y algoritmos
Visibilidad. Métodos **`static`** (IV)

La clase modela cómo deben ser los objetos, pero puede ser algo más:
1.Puede mantener información global sobre si misma o sus instancias.
2.Puede ser un receptor de mensajes.



Atributos **`static`:**
1. *Atributos que se mantienen a nivel de clase, no a nivel de objeto.*

2.Mantienen información sin necesidad de crear un objeto.

Métodos **`static`:**
1.Modelan mensajes cuyo receptor es la clase, no sus objetos.

2.Necesarios cuando se quiere usar un método sin necesidad de crear un objeto.

```
public static double pow(double x,
double y)
```

Se invocan usando el nombre de la clase para referenciar al receptor del mensaje:

```
Math.pow(7.5, 8.4);
```

TEMA 4 : Comunicación y algoritmos
Visibilidad. Métodos **`static`** (V)

Los métodos y campos estáticos tienen sus limitaciones:

→ Un método estático sólo puede acceder a propiedades estáticas del objeto. *Los métodos estáticos sólo pueden:*

- *Invocar métodos estáticos.*
- *Acceder a campos estáticos.*

¿Los métodos estáticos serían independientes de los objetos si no se cumplieran estas restricciones?

¿Se pueden invocar métodos estáticos desde métodos no estáticos?

¿Se puede acceder a atributos no estáticos desde métodos no estáticos?

TEMA 4 : Comunicación y algoritmos
Visibilidad. Métodos **`static`** (VI)

```
class Soldado {

    private static int efectivos;

    public static void enrolar(){
        efectivos++;
    }


    public static void causarBaja(){
        efectivos--;
    }

    public static void combatir(){
        if (efectivos > 10)
            //atacar ..
        else // huir ...
    }
}
```

```
class Calificación {
    public int minimo(int [] n){..}
    public static void main(String args []){
        int [] numeros = { ... }
        int min = minimo(numeros);
        // Error
        // ...
    }
}
```

Sólo hay una variable `efectivos` a la que pueden acceder todos los objetos `soldado`.

Ejemplo `static`: Clase `math`

Métodos	Descripción (valor devuelto)
`abs(x)`	*Valor absoluto. Sobrecargado para int, float y double.*
`ceil(x)`	*Redondeo al entero más pequeño no menor que x*
`cos(x)`	*Coseno (x en radianes)*
`exp(x)`	*x elevado a e.*
`floor(x)`	*Redondeo al entero más grande no mayor que x*
`log(x)`	*Logaritmo natural de x (base e)*
`max(x,y)`	*El mayor de x e y. Sobrecargado para int, float y double.*
`min(x,y)`	*El menor de x e y. Sobrecargado para int, float y double.*
`pow(x,y)`	*x elevado a y*
`sin(x)`	*Seno en radianes*
`sqrt(x)`	*Raíz cuadrada*
`tan(x)`	*Tangente en radianes*
`random()`	*Número double aleatorio entre 0.0 y menor que 1.0*
`round(x)`	*redondea el double x al long más cercano. Versión float a int.*

Referencia `this`

- Supóngase que uno está dentro de un método y que desea conseguir la referencia al objeto actual: para este propósito hay una palabra clave: **`this.`**
- Esta palabra clave -que puede usarse sólo dentro de un método- produce la referencia al objeto por el que se ha invocado al método. Uno puede tratar esta referencia como cualquier otra referencia a un objeto.
- Hay que recordar que si se está invocando a un método de una clase desde dentro de un método de esa misma clase, no es necesario utilizar **`this`**: uno puede simplemente invocar al método.
- Al acceder a variables de instancia de una clase, la palabra clave *this* hace referencia a los miembros de la propia clase.

Referencia this: Ejemplo

```
public class ElementoThis{
    int red;
    int green;
    int black;

    // Contructor de la clase
    public ElementoThis(int red, int green, int black){
    this.red = red; //  la variable local del metodo contructor seria
                           red y this.red hace referencia al atributo de
                           la clase ElementoThis
    this.green = black;
    this.black = black; }

    public int getRed(){
        return this.red; // equivalente a return red;
    }
    .......
}
```

TEMA 4 : Comunicación y algoritmos

Método **main**

- Cuando ejecutamos un programa, el sistema localiza y ejecuta el método ***main()*** de esa clase.
- Al ejecutar la clase se busca el método main que contiene dicha clase:

```
public class Eco {
  public static void main (String[] args){
        System.out.println("hola, soy main");
  }
}
```

- ***main()*** *debe ser* ***public static void*** *y aceptar un único* argumento de tipo ***String[ ].***
  - `String[] args`: parámetros del programa
  - Puede haber más de un `main`, pero sólo se ejecutará uno

TEMA 4 : Comunicación y algoritmos

Modificadores de visibilidad (I)

- **Principio de ocultación de la información:** "A menos que exista una razón importante para que un métodos sea accesible, será declarado privado"

- El objetivo es ocultar en lo posible los detalles de implementación de las clases permitiendo el acceso sólo a través de los métodos públicos

- La ocultación de la información en Java se realiza a través de los **modificadores de visibilidad**

- Los modificadores de visibilidad se aplican a nivel de clase y a nivel de método

TEMA 4 : Comunicación y algoritmos

Modificadores de visibilidad (II)

Los **modificadores de visibilidad** son palabras reservadas que se anteponen a la declaración de los miembros de la clase e indican cómo se puede acceder a dichos miembros desde el exterior.

Distinguiremos:

- **`public`** (público): son accesibles desde todas las clases.
- **`private`** (privado): sólo accesibles desde los métodos de la clase.
- **`protected`** (protegido): Se verán en el próximo tema.

- Sin modificador (por defecto): sólo accesibles desde su Paquete

Por ejemplo:

```
private String titulo;
public void fijarTitulo(String s)
```

```
UnaPersona.fijarTitulo("Dr.")
```

Modificadores de visibilidad (III)

Por respeto al **principio de ocultación de la información**, sólo serán públicos los métodos (nunca los campos, aunque el lenguaje lo permita) que necesariamente deben estar accesibles desde otros objetos.

- Aquellos métodos que sirven de apoyo a los principales incluidos en la especificación se declararán como **`private`.**

- Se construyen métodos que acceden al estado: para cambiarlo o mostrarlo.

Por ejemplo:

```
private String titulo;
public void fijarTitulo(String s)
                { titulo=s;}
```

TEMA 4 : Comunicación y algoritmos

1. Introducción a la programación estructurada
   - Estructuras de repetición en JAVA
   - Entrada de datos en JAVA
2. Comunicación entre objetos
   - Visibilidad. Métodos **`static`**
   - Referencia **`this`**
   - Método **`main`**
   - Modificadores de visibilidad
3. Arrays
   - Unidimensionales
   - Multidimensionales
4. Algoritmos en Arrays
   - Búsquedas
   - Ordenación
   - Análisis de secuencias

TEMA 4 : Comunicación y algoritmos

Arrays: Estructuras estáticas de datos

¿Qué es un array? Es una estructura de datos que almacena una cantidad fija de elementos del mismo tipo, a los cuales se puede acceder por medio de un índice que indica su posición dentro de la estructura.

Es una estructura:

1. **Homogénea**: Todos los datos son de un mismo tipo o clase.

2. **Estática**: Los *arrays contienen siempre el mismo número de datos*
   - En álgebra: vectores (1D), matrices (2D), tensores (3D),...
   - En Java: Son objetos que se declaran, crean, inicializan y consultan.

TEMA 4 : Comunicación y algoritmos

Arrays: Propiedades

- Los arrays se utilizan como contenedores para almacenar datos relacionados (en vez de declarar variables por separado para cada uno de los elementos del array).

- Todos los datos incluidos en el array son del mismo tipo. Se pueden crear arrays de enteros de tipo `int` o de reales de tipo `float`, pero en un mismo array no se pueden mezclar datos de tipo `int` y datos de tipo `float`.

- El tamaño del array se establece cuando se crea el `array` (con el operador `new`, igual que cualquier otro objeto).

- A los elementos del array se accederá a través de la posición que ocupan dentro del conjunto de elementos del `array`.

TEMA 4 : Comunicación y algoritmos

Arrays: Declaración en JAVA (I)

Declaración de un array en **Java:**

`Tipo_de_los_elementos []` *`nombre_de_referencia_al_array`*

O equivalentemente:

`Tipo_de_los_elementos` *`nombre_de_referencia_al_array [];`*

Ejemplos:

```
int [] arrayDeEnteros;          int arrayDeEnteros [];
char [] arrayDeCaracteres;      char arrayDeCaracteres [];
```

En la declaración no se especifica el tamaño del array.

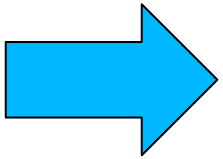
- *¿Por qué? → Porque en la declaración no se crea el array, sino la referencia que lo apunta.*
- *De hecho, lo que realmente estamos declarando es una referencia al array.*

TEMA 4 : Comunicación y algoritmos

Arrays: Declaración en JAVA (II)

Declaración de arrays:

```
int [] arrayDeEnteros;
char [] arrayDeCaracteres;
```



¡Ojo!: ***Estos ejemplos declaran una referencia al array***
Todavía no tengo ningún array, sino algo que puede apuntar (referirse) a un array.

Los arrays en Java son objetos y a los objetos en Java no se accede *nunca de forma directa, sino a través de referencias.*

La declaración **`int [] serie`** **significa que:**

- Obtengo una referencia llamada **serie que puede apuntar a un array de enteros,** pero de momento, dependiendo de dónde se declare:
  - → *o bien no apunta a nada*
  - → *o bien apunta a algo indefinido*
- Para que **serie apunte a un *array debemos crear dicho array.***

TEMA 4 : Comunicación y algoritmos

Arrays: Creación en JAVA (I)

Crear un array es asignarle un espacio de almacenamiento en memoria.

Para ello, hay que usar el operador **new**, seguido por:

- *El tipo de los elementos del array*
- *El número de elementos que va a contener entre corchetes*

```
new tipo_elemento [n_elementos]
```

- El operador **new** crea un ***array*** de ***n_elementos*** del tipo ***tipo_elemento*** y devuelve una referencia al mismo.
- *La creación del array fija su número de elementos, que permanece inmutable:*

```
new int[5]; // Crea un array de 5 enteros
            // y devuelve una referencia al mismo.
```

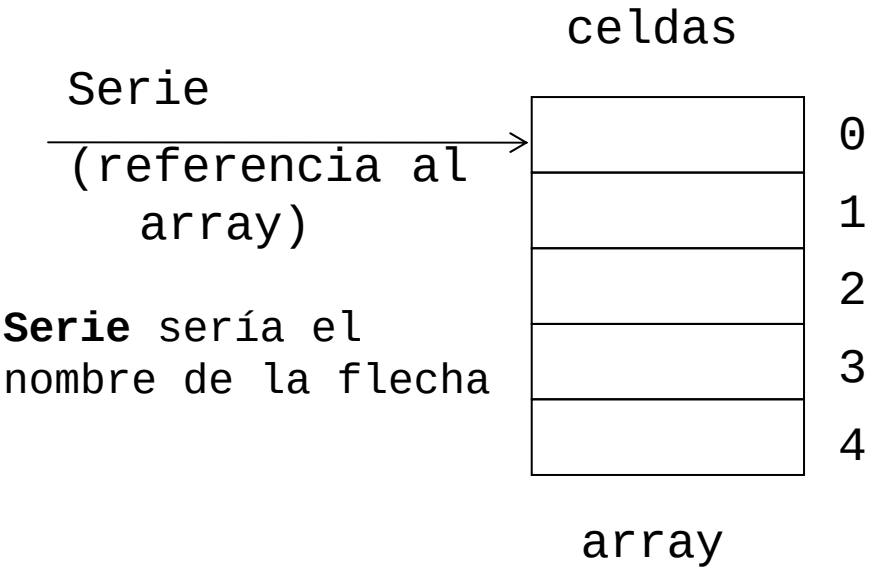
TEMA 4 : Comunicación y algoritmos

Arrays: Creación en JAVA (II)

Una vez creado un array, ¿Cómo lo usamos?

Asignando la referencia que devuelve new a una referencia a un array.

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



Serie sería el nombre de la flecha

Posiciones del array (ocupan posiciones contiguas)

TEMA 4 : Comunicación y algoritmos

Arrays: Creación en JAVA (III)

Los arrays se pueden declarar y crear simultáneamente:

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

Se pueden declarar y crear varios arrays que contengan el mismo tipo de elementos en una sola línea:

```
int a[] = new int[20], b[] = new int[100];
```

TEMA 4 : Comunicación y algoritmos

Arrays: objetos y referencias (I)

A los objetos se accede mediante referencias que *apuntan a los mismos.*

→ Los *arrays en* Java son objetos.

Ejemplo: Una referencia es a un objeto lo que la dirección a un domicilio.

La dirección de tu casa ≠ Tu Casa

Referencia ≠ Objeto

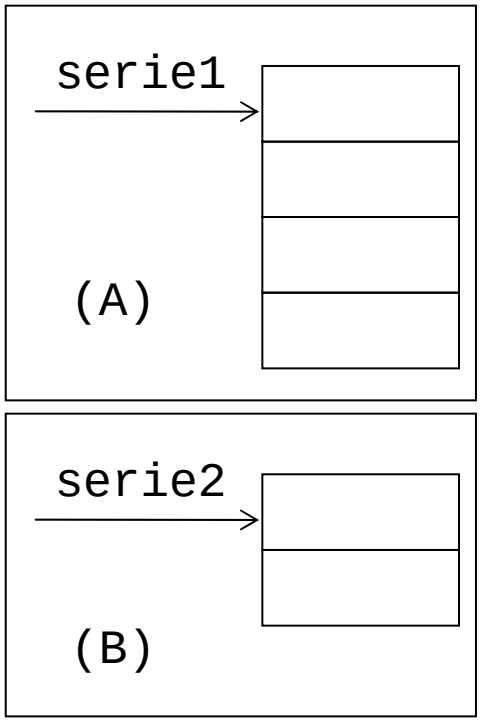
Ambos son conceptos relacionados: Una carta no llega a su destino si no pones bien las señas.

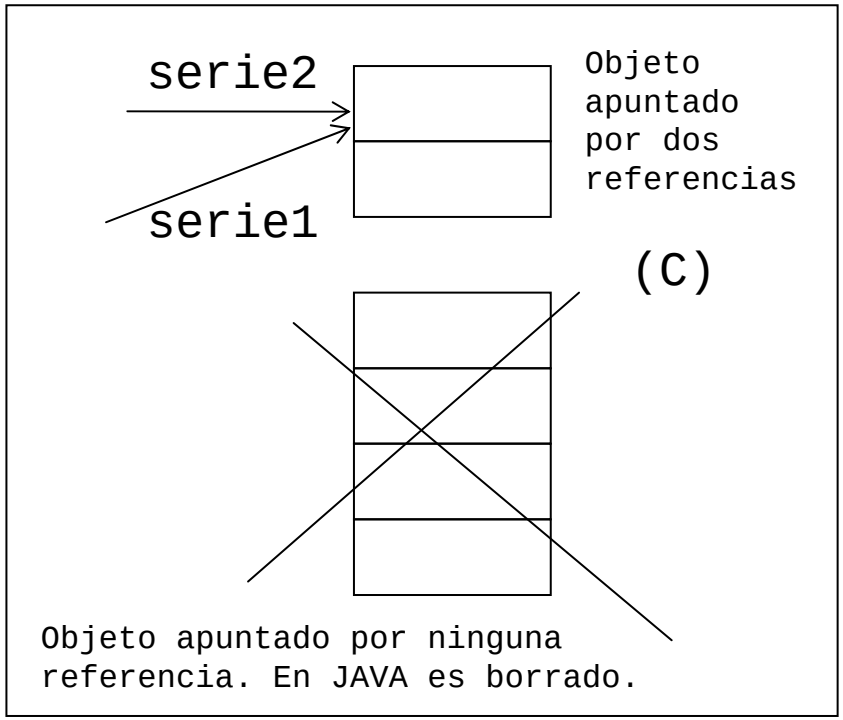
¿Qué significa esto en el ámbito de los arrays?

TEMA 4 : Comunicación y algoritmos

Arrays: objetos y referencias (II)

```
(A)    int [] serie1 = new int[5];
(B)    int [] serie2 = new int[3];
(C)    serie1 = serie2;
```

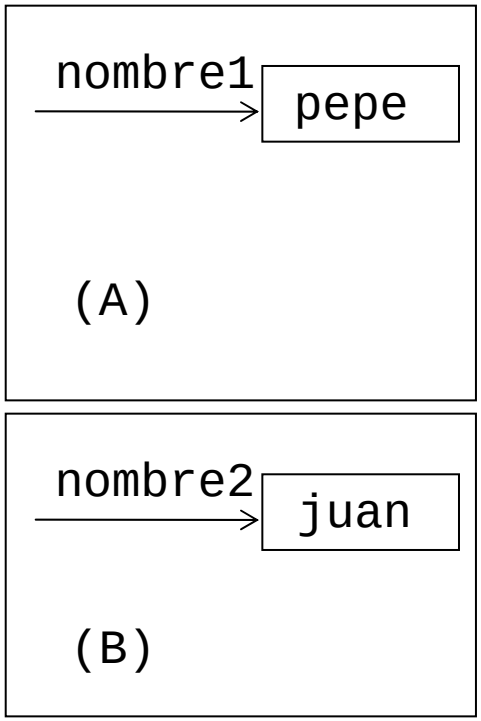


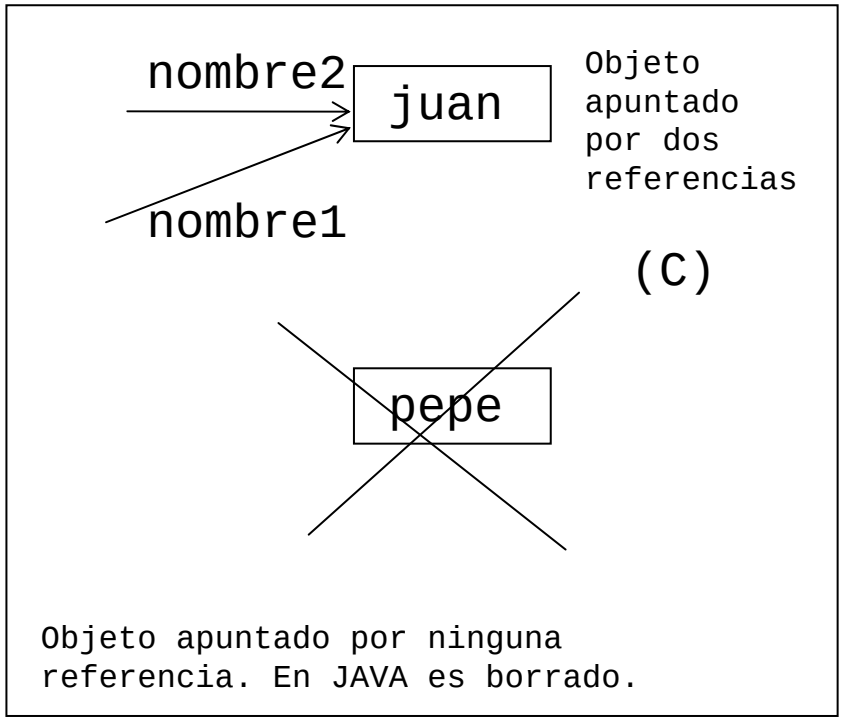


TEMA 4 : Comunicación y algoritmos

Objetos y referencias (III)

```
(A)    String nombre1 = “pepe”;
(B)    String nombre2 = “juan”;
(C)    nombre1 = nombre2;
```





TEMA 4 : Comunicación y algoritmos

Arrays: inicialización (I)

Con la declaración **`int [] serie;`**

Se tiene la referencia, pero todavía ningún array.

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

Se tiene un array de 5 enteros apuntado por la referencia ***serie***

¿Cuánto valen los enteros del *array referenciado por* ***serie?***

Dos posibilidades:

→ *O bien valen* ***0*** (si serie ha sido declarada a nivel de clase).

→ *O bien su valor es indefinido (si* ***serie*** ha sido declarada dentro de un *método).*

TEMA 4 : Comunicación y algoritmos

Arrays: inicialización (II)

Valores iniciales por defecto: Cuando un *array se declara como variable de* instancia, al crearlo sus elementos se inicializan de forma automática a:

- *Cero, en el caso de* ***tipos primitivos numéricos.***
- *Carácter nulo en el caso de* ***char***
- **false** en el caso de boolean
- **null** en el caso de referencias

TEMA 4 : Comunicación y algoritmos

Arrays: inicialización (III)

La declaración, creación e inicialización de un array son operaciones distintas, que pueden realizarse de forma independiente:

```
int [] serie; // Declaración.
serie = new int[20]; // Creación.
int serie2 [] = new int[100] // Declaración y creación
```

Puede hacerse en el momento de la declaración, continuando con:

→ *Un signo igual y*

→ *Una lista separada por comas y encerrada entre llaves de inicializadores.*

```
int [] serie = {1,2,3,4,5}; //Se ha creado implícitamente
```

El tamaño del *array queda determinado en este caso por el número de* elementos de la lista.

TEMA 4 : Comunicación y algoritmos

Arrays: inicialización (IV)

Las inicializaciones del estilo:

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

Son muy útiles en algunos casos, pero:

- ¿Y si necesitamos un *array con 5000 elementos?*
- ¿Y si nos interesa que todos los valores del *array se inicialicen al* mismo valor?

Necesitamos acceder de forma sencilla y eficiente a cada una de las posiciones del array. ¿Cómo? Mediante índices

TEMA 4 : Comunicación y algoritmos

Arrays: indexación de elementos (I)

Nos podemos referir a cualquier elemento del array mediante el nombre del array seguido de la posición del elemento entre corchetes.

Consideremos la sentencia:

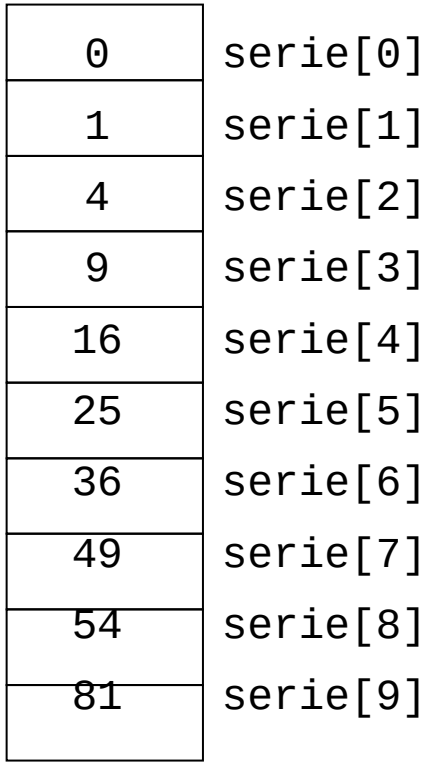
```
int [] serie = new int[10];
```

y la siguiente porción de código que inicializa los elementos del *array:*

```
for (int i = 0; i < serie.length; i++) {
        serie[i] = i * i;
}
```

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

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



TEMA 4 : Comunicación y algoritmos

Arrays: indexación de elementos (II)

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

Ejemplo:

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

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

Ejemplo:

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

TEMA 4 : Comunicación y algoritmos

Longitud de arrays: atributo **`lenght`** (I)

Consideremos otra vez el ejemplo anterior:

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

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

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

No debemos acceder a posiciones del array inexistentes:

- Si accedemos se lanzará una excepción.

TEMA 4 : Comunicación y algoritmos

Longitud de arrays: atributo **lenght** (II)

Los arrays son objetos. Los objetos tienen atributos que pueden consultarse si son públicos. Sería de ayuda poder conocer de un array su longitud.

Java define el atributo `length` que guarda el número de elementos del `array` *y* que se consulta según la sintaxis:

```
nombre_array.length
```

Ejemplo:

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

Cuando procesamos un array hay que tener precaución:

- El subíndice no puede sobrepasar la posición (length –1).
- El subíndice no puede ser menor que cero.

TEMA 4 : Comunicación y algoritmos

Longitud de arrays: atributo **lenght** (III)

Ejemplo: inicialización y suma de elementos de un array

```
class Ejemplo1{
   final static int TAMAÑO_ARRAY = 10;

   public static void main(String args[]){
        int s[];
       int total;
        s = new int[TAMAÑO_ARRAY];
        for (int i = 0; i < s.length; i++) {
            s[i] = 2 + 2 * i; // 2, 4, 6, ..., 20
       }
       // suma elementos
       for ( int i= 0; i < s.length; i ++) {
          total += s[i];
       }
} // Fin de ejemplo 1
```

TEMA 4 : Comunicación y algoritmos

Arrays: algunas observaciones (I)

- La palabra array se ha traducido de muchas maneras, sin que ninguna de ellas haya cuajado por completo. Podemos encontrar en español: arreglos, formaciones, matrices, vectores.

- Los arrays son objetos y por tanto accedemos a ellos usando referencias.

  → Sin embargo, no existe una clase array ni métodos asociados a los arrays. Los arrays son una construcción del lenguaje.

- La expresión `serie.length` no es una invocación de un método, sino la lectura de un dato asociado al array (su tamaño).

TEMA 4 : Comunicación y algoritmos

Arrays: algunas observaciones (II)

- No hay restricciones respecto del tipo de datos que puede contener un array.

  → Pueden ser tanto tipos de datos primitivos como referencias a objetos de una clase.

- Los arrays tienen un tamaño fijo:

  → No pueden crecer ni disminuir su tamaño.

  → Se dice que son estructuras estáticas de datos (en contraposición con las estructuras dinámicas de datos que si pueden variar el tamaño).

- No obstante, las referencias de arrays se pueden asignar a arrays de diferentes tamaños.

TEMA 4 : Comunicación y algoritmos

Arrays: estructuras multidimensionales (I)

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

- El primer subíndice se refiere a la fila (es el índice del array de arrays)
- El segundo subíndice se refiere a la columna (es el índice de cada uno de los arrays del array de arrays)

Un array de dos dimensiones de m filas y n columnas es un array **m x n.**

Ejemplo: En la tabla se muestra la disposición de los elementos de un array a de **3x4:** `int [][] a = new int[3][4];`

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

- **a** es un array de 3x4 enteros
- **a[i]** es un array de 4 enteros
- **a[i][j]** es un entero

TEMA 4 : Comunicación y algoritmos

Arrays: estructuras multidimensionales (II)

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

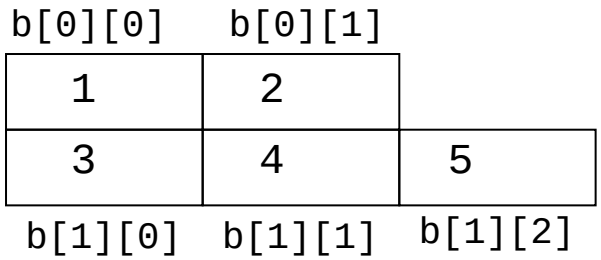
Ejemplo: *Array con doble subíndice*

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

Los valores se agrupan por fila encerrados entre llaves:

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

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



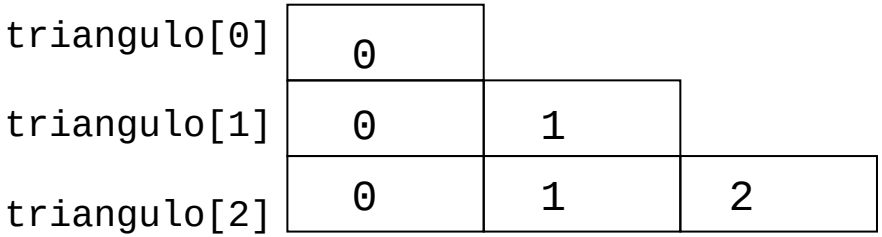
Lo que estamos declarando es un *array cuyos elementos son dos arrays, uno* conteniendo dos elementos **{1,2}** y el otro conteniendo tres elementos **{3,4,5}.**

TEMA 4 : Comunicación y algoritmos

Arrays: estructuras multidimensionales (III)

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

```
int [][] triangulo = new int [3][];
triangulo[0] = new int[1];
triangulo[1] = new int[2];
triangulo[2] = new int[3];
triangulo[0][0] = 0;
triangulo[1][0] = 0;
triangulo[1][1] = 1;
triangulo[2][0] = 0;
triangulo[2][1] = 1;
triangulo[2][2] = 2; O bien,
int[][] triangulo = { {0} ,{0,1} ,{0,1,2} };
```



¡ Triangulo tiene dos dimensiones pero no es una matriz !

TEMA 4 : Comunicación y algoritmos

Arrays: estructuras multidimensionales (IV)

Ejemplo: Suma de los elementos de un *array bidimensional:*

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

    public static void main (String args[] ) {
        for (int fila = 0; fila < a.length; fila++)
            for (int col = 0; col < a[fila].length; col++)
            total += a[fila][col];
        }
    }
```

// a.length: ***num. de filas (elementos del array a)***
// a[fila].length: ***num. de columnas (elementos del array a[fila])***

TEMA 4 : Comunicación y algoritmos

Arrays: estructuras multidimensionales (V)

- **Ejemplo:** Calcula la nota máxima y mínima de un array de calificaciones, donde cada fila representa un estudiante y cada columna representa una calificación en diferentes exámenes

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

   public static void main (String args[]){
    int nota_baja = 100, nota_alta = 0;

    for (int i = 0; i < notas.length; i++ )
        for (int j = 0; j < notas[i].length; j++ )
            if (notas[i][j] < nota_baja) nota_baja = notas[i][j];
    System.out.println("La nota baja es" + nota_baja);

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

TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays

- Importancia en organizar la Información

- Ej: Búsqueda de un número de teléfono en la guía
  - Rápido, gracias a que está ordenado alfabéticamente

- En informática, frecuentemente es necesario buscar información entre mucha disponible, para lo que también se ordena.
- Existen diversos algoritmos para ordenar una lista de elementos y para buscar dentro de una lista ordenada

TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsquedas

Búsqueda en arrays: *Se trata de determinar si el array contiene un elemento que coincide con el valor de una clave.*

→ *La clave de búsqueda.*

→ *Se denomina búsqueda al proceso de encontrar ese elemento clave / su posición.*

•Es una operación muy frecuente en programación, por lo que existen diversidad de algoritmos

- Con diferencias de eficiencia entre ellos

•Diferenciaremos dos tipos de búsqueda:

- **Secuencial** y **Dicotómica**

TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda secuencial (I)

Aplicabilidad:

- Desconocimiento acerca de la organización de los datos
- Estructura sólo accedida secuencialmente

Idea clave:

- Visitar todas las posiciones del array, hasta que se encuentre el elemento o se llegue al final del array (el elemento no está).

TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda secuencial (II)

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

- Si el *array no está ordenado es igualmente* probable que el elemento buscado esté al principio o al final.
- El número medio de comparaciones es igual a la mitad del tamaño del array.
- En el peor de los casos: el tamaño del array

TEMA 4 : Comunicación y algoritmos

Búsquedas en arrays: Código secuencial

```
public static int secuencial(int a[], int elem) {
   int i;  boolean encontrado;
   i=0; encontrado= false;
   while (i<=a.length-1 && !encontrado)
                if (elem == a[i])
                    encontrado = true;
                else
                    i++;
                if (encontrado)
                    return i;
                else
                    return i-1; //no se ha encontrado
}
```

TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda dicotómica (I)

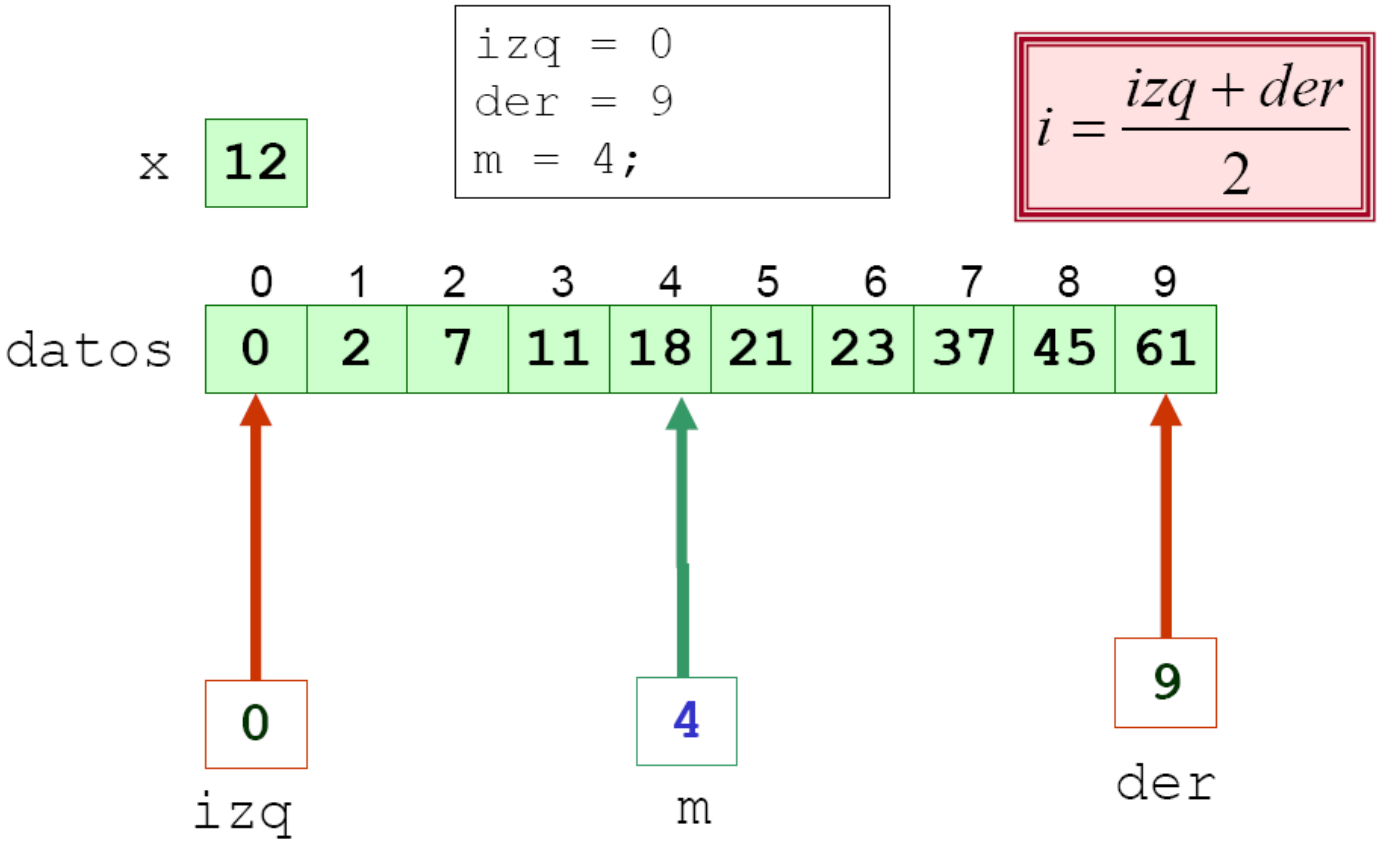
Aplicabilidad: El array debe estar ordenado

Ejecución:

- Se mira el elemento del centro de la lista
- Si es el buscado, se devuelve
- Si es mayor que el buscado, se repite la búsqueda sobre la primera mitad de la lista
- Si es menor que el buscado, se repite la búsqueda sobre la segunda mitad de la lista
- Se repite el algoritmo hasta que:
  - La clave es igual al elemento medio del subarray restante
  - El subarray restante consta de un solo elementos distinto a la clave

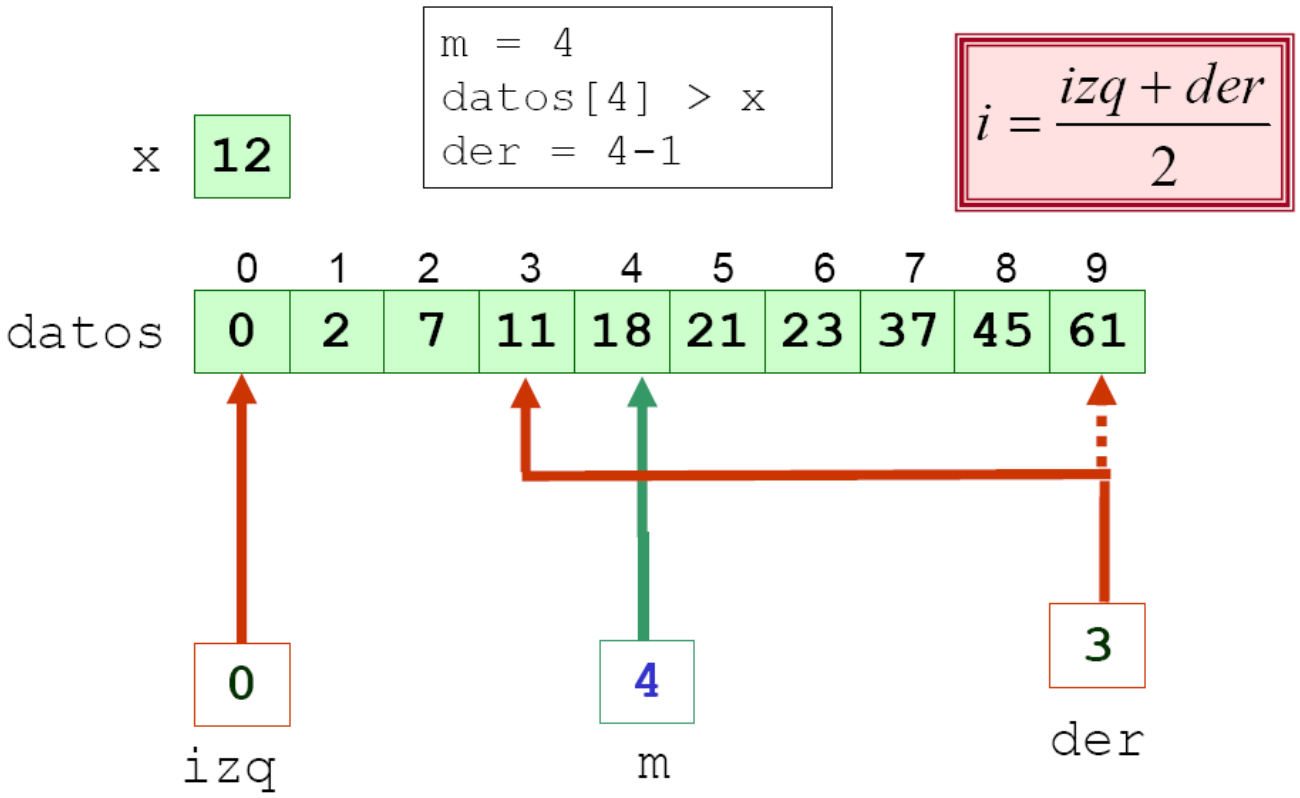
TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda dicotómica (II)



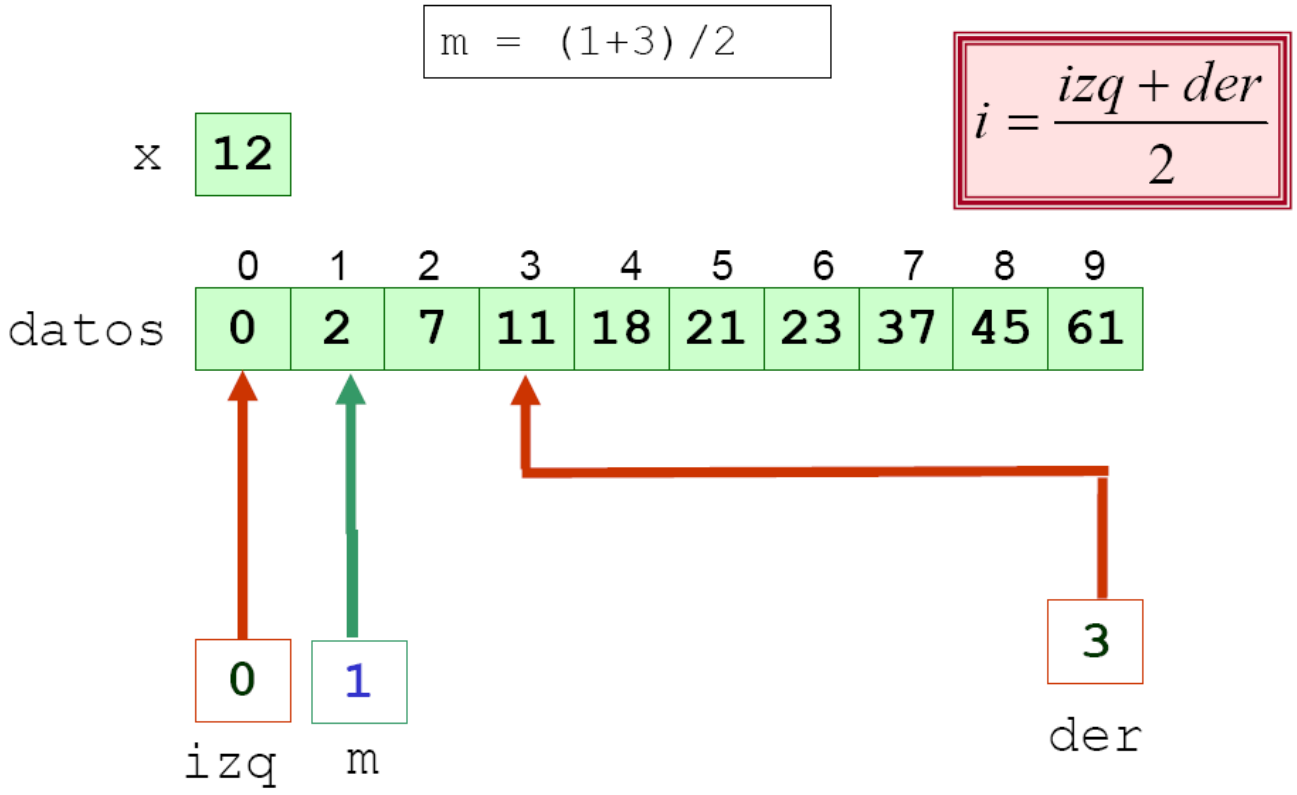
TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda dicotómica (III)



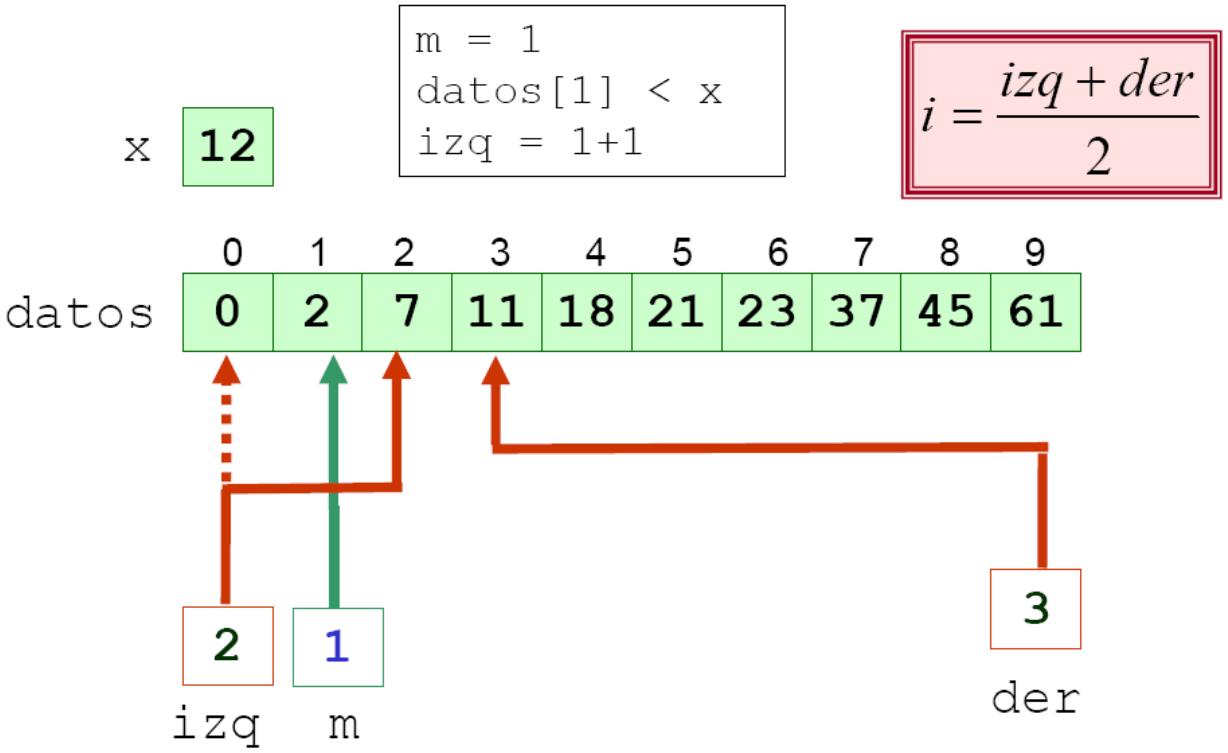
TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda dicotómica (IV)



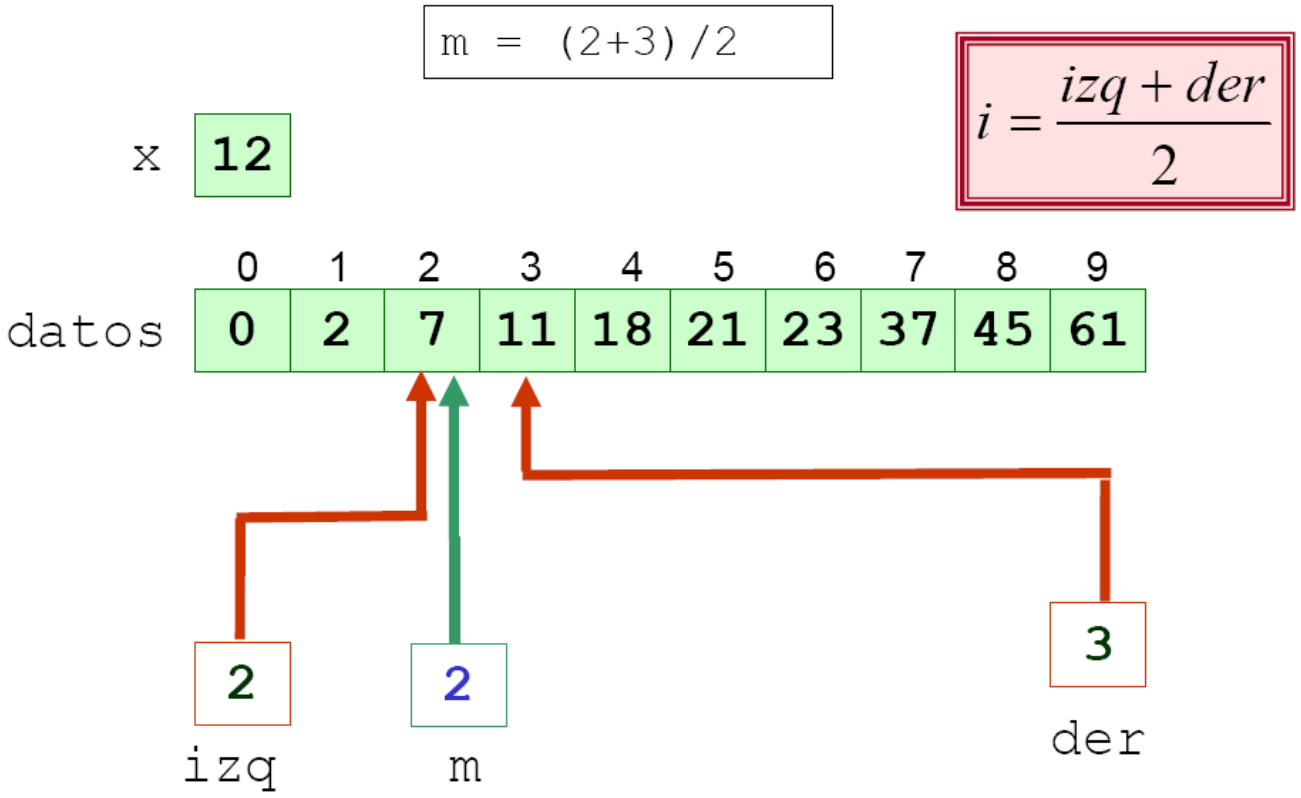
TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda dicotómica (V)



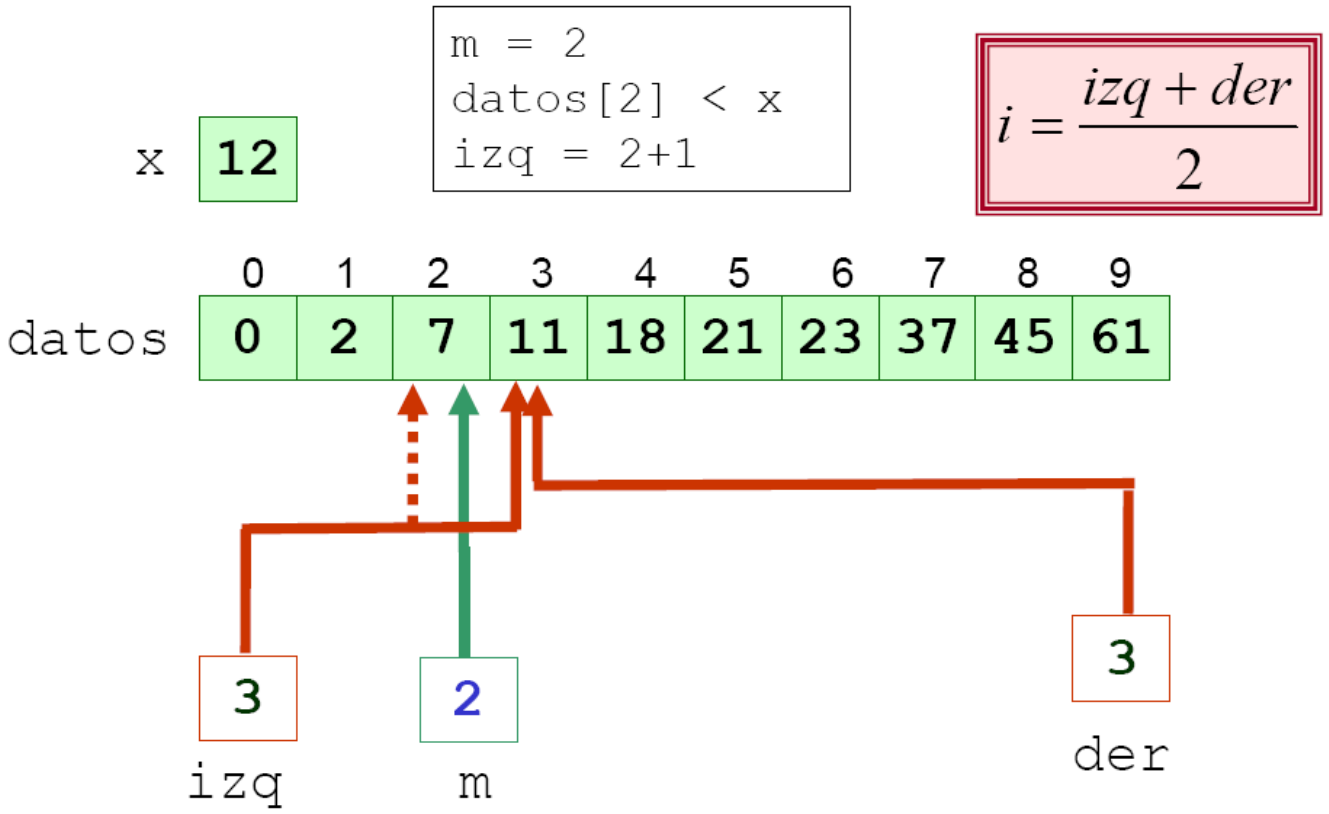
TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda dicotómica (VI)



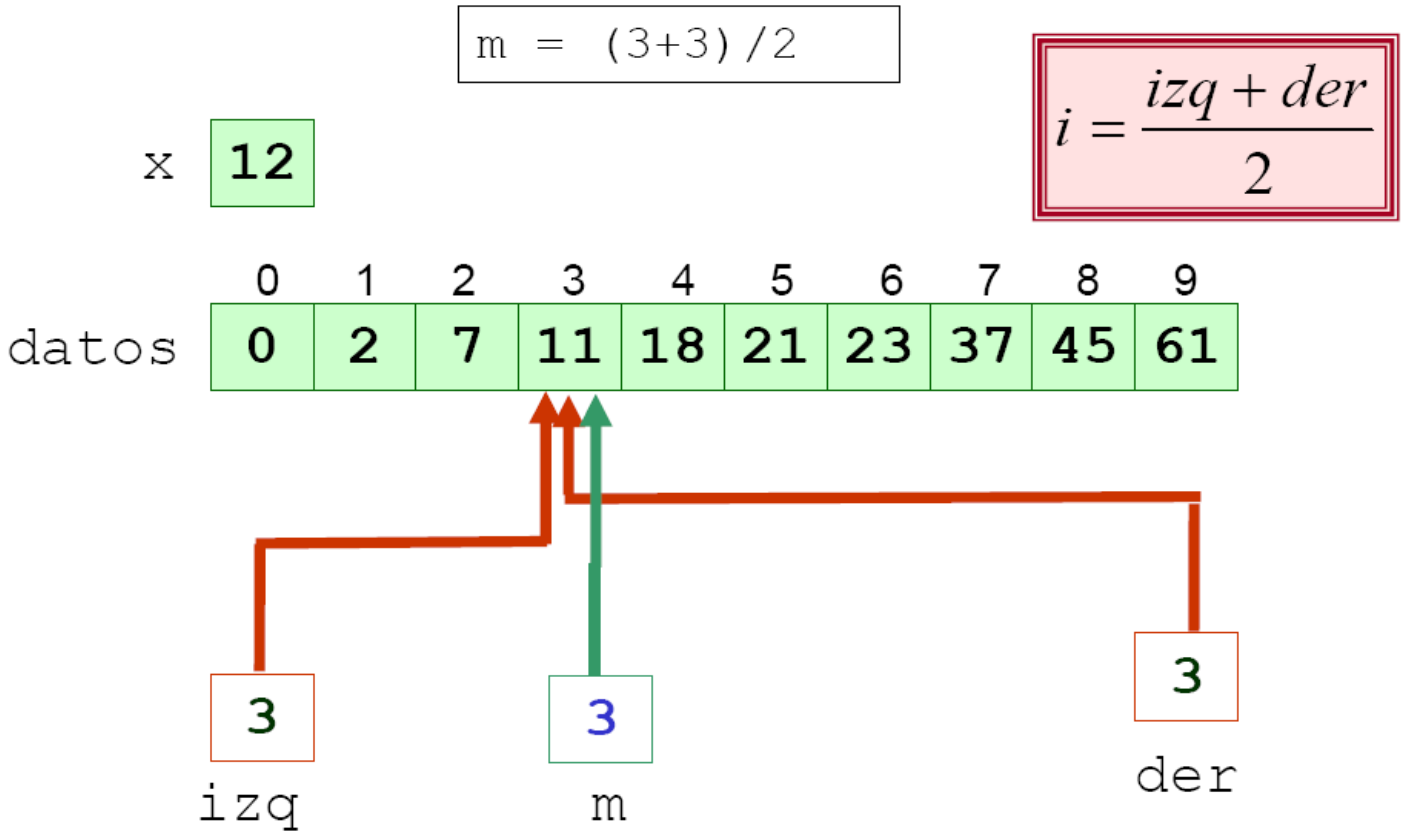
TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda dicotómica (VII)



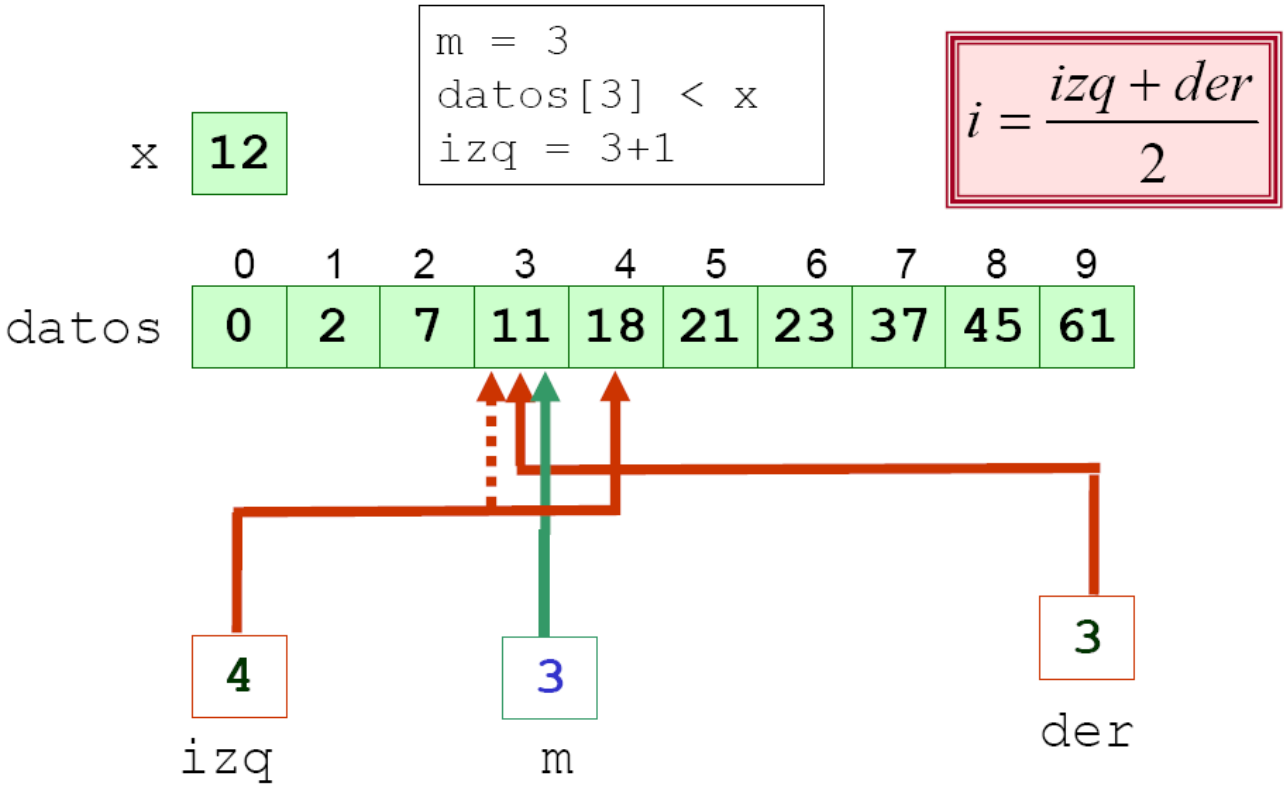
TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda dicotómica (VIII)



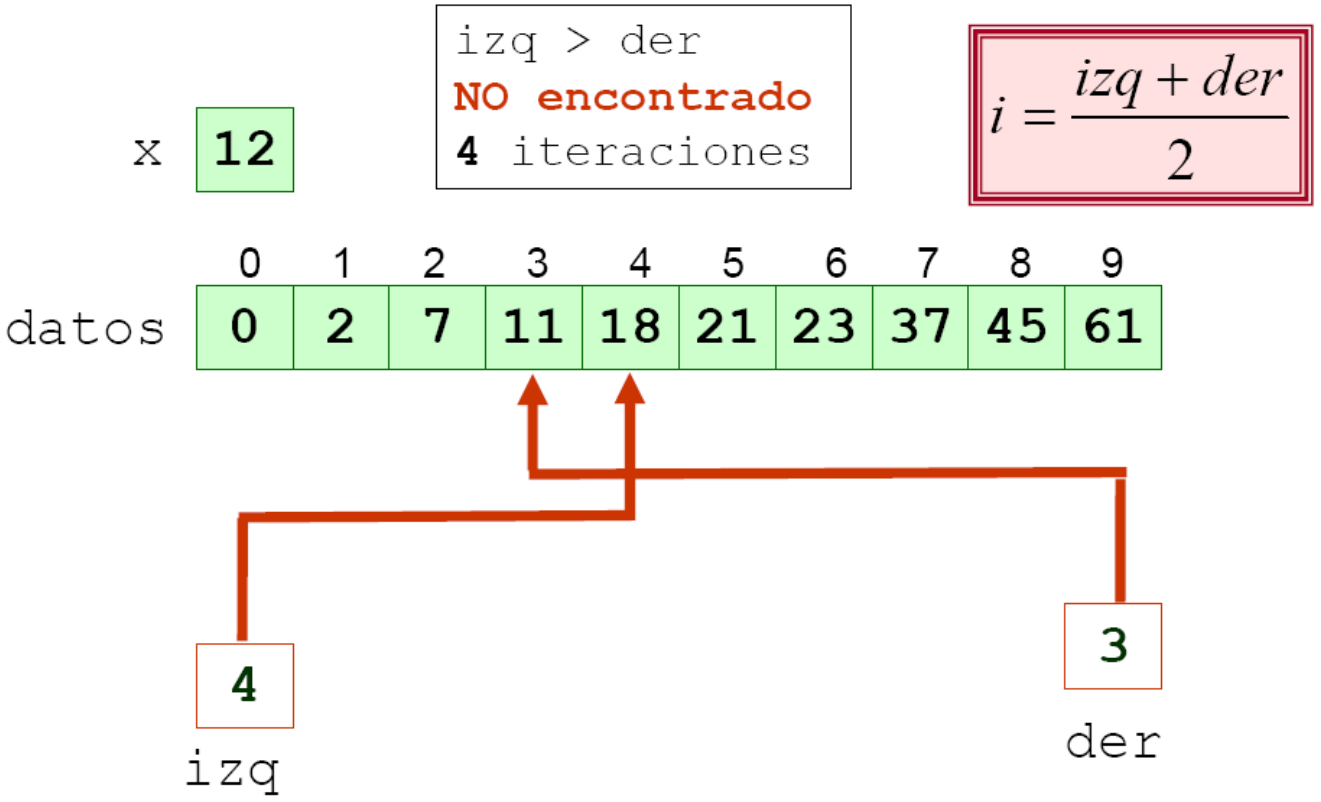
TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda dicotómica (IX)



TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Búsqueda dicotómica (X)



TEMA 4 : Comunicación y algoritmos

Búsquedas en arrays: Código dicotómica

```
public static int binaria(int a[], int elem, int posicion) {
   int i=0, menor =0, mayor= a.length-1, medio=0;
   boolean encontrado =false;;
   while (!encontrado && mayor >= menor){
        medio = (mayor + menor)/2;
                if (elem == a[medio]) encontrado = true;
                else if (elem > a[medio]) //buscar en la parte superior
                    menor = medio + 1;
                else                      //buscar en la parte inferior
                     mayor = medio - 1;
   }
   if (encontrado) posicion = medio;
   else posicion = menor;
}
```

TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Eficiencia (I)

Estudio de la eficiencia

en la peor situación: si el elemento no se está en el array

- Búsqueda Secuencial: El número de comparaciones es

  `n= a.length`

- Búsqueda Binaria: En cada comparación se puede eliminar la mitad de los elementos del array.

  Primera comparación: array de n elementos

  Segunda comparación: array de n/2 elementos

  Tercera comparación: array de n/4 elementos

  Cuarta comparación: array de n/8 elementos

  ... ...

  C-ésima comparación: array de n/ 2^c elementos

TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Eficiencia (II)

Búsqueda Binaria: C-ésima comparación: array de n/ 2^c elementos

Para llegar a que el array tenga 1 sólo elemento:

$n/2^c = 1$ por tanto $n = 2^c$ por lo que $C = \log2(n)$

n	Número de Comparaciones	
	Búsqu. secuencial	Búsqu. binaria
10	10	4
100	100	7
1000	1000	10
10000	10000	14
100000	100000	17
1000000	1000000	20

TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Ordenación

- La ordenación de datos es una tarea relevante en programación
- Es un problema clásico ampliamente estudiado, por lo que existen diversidad de algoritmos

- Los algoritmos difieren en su eficiencia:
  1. Economía en el uso de memoria
  2. Economía en el número de operaciones:
     - Algoritmos directos: **Inserción**, **Selección** e **Intercambio** (o Burbuja)

TEMA 4 : Comunicación y algoritmos

Algoritmos en arrays: Ordenación (I)

- Consideramos un array de datos simples o complejos
- Sobre el array debe haber definida una relación de orden

- **El problema de la ordenación** (ascendente) de un array **a[]**, consiste en encontrar una permutación **s** de los índices del array, tal que se cumpla:

$\forall$ ***pareja de subíndices i,j, si*** $i \leq j \Rightarrow a[i] \leq a[j]$

TEMA 4 : Comunicación y algoritmos

Ordenación en arrays: intercambio directo (I)

Idea clave: La idea básica del ordenamiento de un array con éste método:

- Está basado en intercambio de pares adyacentes
- Se hacen varias pasadas sobre el array comparando todos los elementos adyacentes, intercambiándolos si no están ordenados
- En el peor caso se realizan **`(a.lenght –1)`** pasadas sobre el array

- Se conoce popularmente como el **algoritmo de la burbuja**
- **Ejemplo:** Se ha de ordenar la siguiente colección

TEMA 4 : Comunicación y algoritmos

Ordenación en arrays: intercambio directo (II)

Un array está ordenado si:

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

Algoritmo de ordenación de la burbuja:

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

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

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

TEMA 4 : Comunicación y algoritmos

Ordenación en arrays: Burbuja (I)

- Paso 1: primera pasada

72	64	50	23	85	18	45	8
64	72	50	23	85	18	45	8
64	50	72	23	85	18	45	8
64	50	23	72	85	18	45	8
64	50	23	72	85	18	45	8
64	50	23	72	18	85	45	8
64	50	23	72	18	45	85	8
64	50	23	72	18	45	8	85

Ordenación en arrays: Burbuja (I)

- Paso 2: segunda pasada

64	50	23	72	18	45	8	85
50	64	23	72	18	45	8	85
50	23	64	72	18	45	8	85
50	23	64	72	18	45	8	85
50	23	64	18	72	45	8	85
50	23	64	18	45	72	8	85
50	23	64	18	45	8	72	85
50	23	64	18	45	8	72	85

TEMA 4 : Comunicación y algoritmos

Ordenación en arrays: Burbuja (II)

- Paso 3: tercera pasada

50	23	64	18	45	8	72	85
23	50	64	18	45	8	72	85
23	50	64	18	45	8	72	85
23	50	18	64	45	8	72	85
23	50	18	45	64	8	72	85
23	50	18	45	8	64	72	85
23	50	18	45	8	64	72	85
23	50	18	45	8	64	72	85

TEMA 4 : Comunicación y algoritmos

Ordenación en arrays: Burbuja (III)

- Paso 4: cuarta pasada

23	50	18	45	8	64	72	85
23	50	18	45	8	64	72	85
23	18	50	45	8	64	72	85
23	18	45	50	8	64	72	85
23	18	45	8	50	64	72	85
23	18	45	8	50	64	72	85
23	18	45	8	50	64	72	85
23	18	45	8	50	64	72	85

TEMA 4 : Comunicación y algoritmos

Ordenación en arrays: Burbuja (IV)

- Paso 1: quinta pasada

23	18	45	8	50	64	72	85
18	23	45	8	50	64	72	85
18	23	45	8	50	64	72	85
18	23	8	45	50	64	72	85
18	23	8	45	50	64	72	85
18	23	8	45	50	64	72	85
18	23	8	45	50	64	72	85
18	23	8	45	50	64	72	85

TEMA 4 : Comunicación y algoritmos

Ordenación en arrays: Burbuja (V)

- Paso 6: sexta pasada

18	23	8	45	50	64	72	85
18	23	8	45	50	64	72	85
18	8	23	45	50	64	72	85
18	8	23	45	50	64	72	85
18	8	23	45	50	64	72	85
18	8	23	45	50	64	72	85
18	8	23	45	50	64	72	85
18	8	23	45	50	64	72	85

TEMA 4 : Comunicación y algoritmos

Ordenación en arrays: Burbuja (VI)

- Paso 7: séptima pasada

18	8	23	45	50	64	72	85
8	18	23	45	50	64	72	85
8	18	23	45	50	64	72	85
8	18	23	45	50	64	72	85
8	18	23	45	50	64	72	85
8	18	23	45	50	64	72	85
8	18	23	45	50	64	72	85
8	18	23	45	50	64	72	85

TEMA 4 : Comunicación y algoritmos

Ordenación en arrays: Código Burbuja

```
public static void burbuja(int a[] ) {
   int aux;
   for (int i = 0; i < a.length-1; i++ )
        for (int j = 0 ; j < a.length- 1- i; j++)
            if (a[j] > a[j+1]){
                aux = a[j];
                a[j] = a[j+1];
                a[j+1] = aux;
   }
}
```

TEMA 4 : Comunicación y algoritmos

Algoritmos: análisis de secuencias (I)

- Problema: analizar secuencias de datos.
- Por ejemplo:
  - cuántos números pares hay en la secuencia

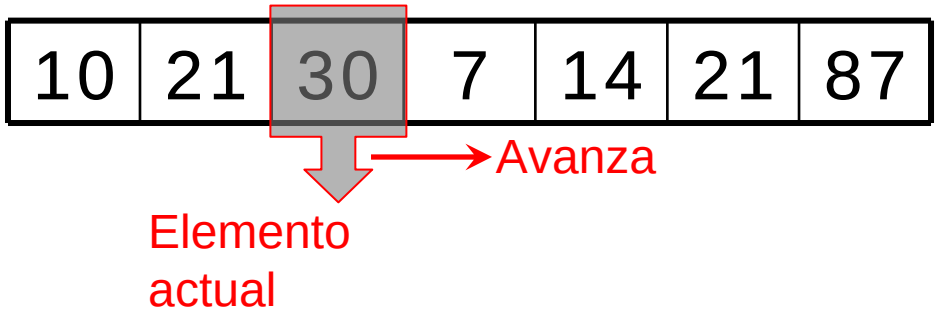
    11 12 7 13 15 18 21
  - aparece la cadena “co” en

    mi **co**che es de **co**lor rojo
  - y cuántas veces aparece

TEMA 4 : Comunicación y algoritmos

Algoritmos: análisis de secuencias (II)

- Una secuencia es una lista de datos en la que:
  - solo podemos acceder a un elemento (elemento actual);
  - y solo podemos avanzar a por el siguiente elemento de la secuencia.



TEMA 4 : Comunicación y algoritmos

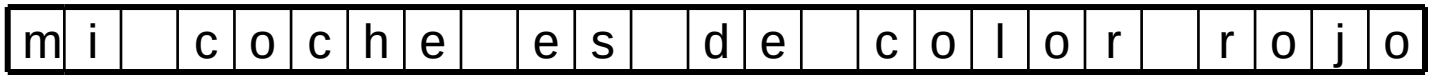
Algoritmos: identificar elementos en secuencias

- El primer paso (y más sencillo) para analizar una secuencia es identificar los elementos de la secuencia.
- Ejemplo: si tenemos como entrada

  mi coche es de color rojo

  si nos piden

  –Hallar si se encuentra la subcadena “co”. Secuencia analizar:



  –Cuántas palabras hay detrás de la primera aparición de “coche”. Secuencia analizar

Leer ficheros de texto

- Muchas de las secuencias que estudiemos estarán en un fichero de texto.

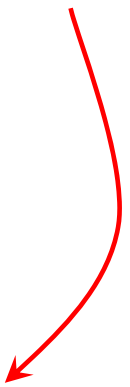
- Para leer de un fichero se usa la clase ***File*** del paquete java.io junto con un Scanner

- Si el fichero que queremos leer no existe o no tenemos permiso para leerlo, se lanza una excepción que hay que capturar.

Leer ficheros de texto: Ejemplo (1)

- Leer palabras del fichero *fichero.txt* y mostrarlas por pantalla

```
import java.io.*; //para File
import java.util.*; //para Scanner
. . . . . .
Scanner scan;
try {
   scan = new Scanner(new File(“fichero.txt”));
   //mientras haya palabras en el fichero
   while(scan.hasNext()) {
      String sig = scan.next();
      System.out.println(next);
   }
} catch (Exception e) {
   System.out.println(“El fichero no existe”);
}
. . . . .
```



Si fichero no existe, salta a catch

Leer ficheros de texto: Ejemplo (2)

- Leer enteros del fichero *fichero.txt* y mostrarlos por pantalla

```
import java.io.*; //para File
import java.util.*; //para Scanner
. . . . . .
Scanner scan;
try {
   scan = new Scanner(new File(“fichero.txt”));
   //mientras haya enteros en el fichero
   while(scan.hasNextInt()) {
      int sig = scan.nextInt();
      System.out.println(next);
   }
} catch (Exception e) {
   System.out.println(“El fichero no existe”);
}
. . . . .
```

Leer caracter a caracter

- No existe un método de Scanner para leer carácter a carácter. **No existe** un método *char nextChar()*
- Solución: cambiar delimitadores por defecto de un Scanner.

```
//contar número de aes. Suponemos que tenemos creado
//una variable de tipo Scanner

scan.useDelimiter(“”); //delimitadores fuera
int contador = 0;
while (scan.hasNext()) {
   String s = scan.next();//s tiene un único car
   if (s.equals(“a”)) { //no se puede s == “a”
       contador++;
   }
}
```

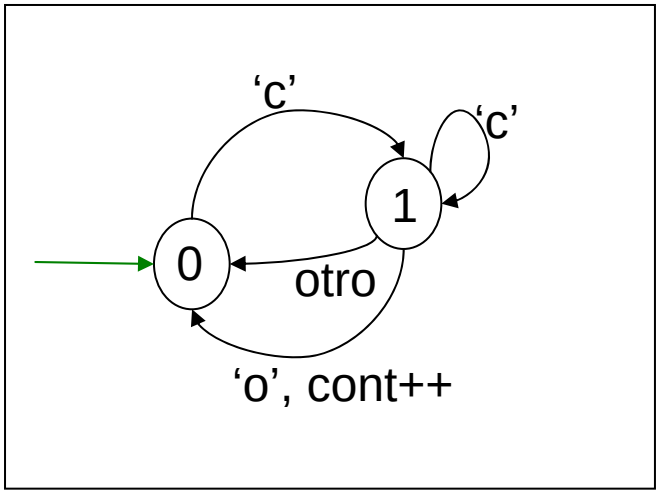
Máquina de estados (I)

- Para resolver un problema de análisis de secuencias, nos puede ayudar *"pintar"* el problema mediante una *máquina de estados*.

- Una máquina de estados es un grafo compuesto de
  - vértices etiquetados por números enteros: 0, 1, 2,…. Son los *estados* de la máquina.
  - aristas entre vértices etiquetadas por una condición y, opcionalmente, acciones: indican qué condición ha de cumplirse para que la máquina cambie de estado y qué acciones se han de llevar a cabo cuando se cambia de estado.

Máquina de estados: Ejemplo (1)

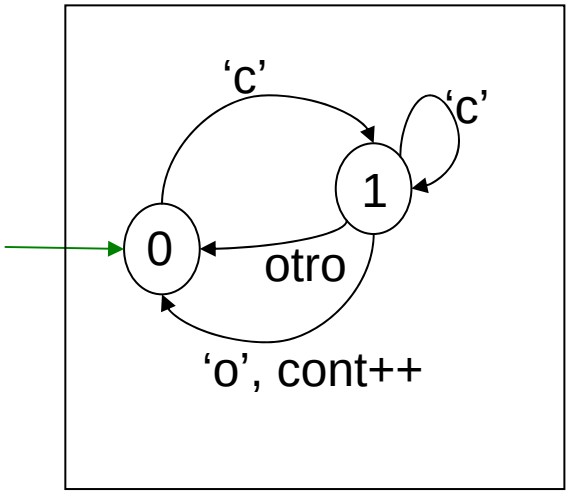
Contar apariciones de la subcadena “co”



- Se empieza por la arista que no tiene estado asociado a su inicio (arista verde)
- En estado 0:
  - Si elemento actual es igual a ‘c’, se cambia al estado 1. ***Hemos encontrado una ‘c’***.
  - Si no, se permanece en estado 0. ***Todavía no tenemos ‘c’*.**
- En estado 1:
  - si elemento actual es ‘c’, permanecer en estado 1. ***Seguimos con una ‘c’***
  - si es igual a ‘o’, aumentar contador y pasar a estado 0. ***Hemos encontrado una ‘o’ después de ‘c’*.**
  - si distinto de ‘c’ , ‘o’, pasar a estado 0. ***Volvemos a estar sin ‘c’***

Máquina de estados: Ejemplo (2)

Contar apariciones de la subcadena “co” en el fichero “texto.txt”




```
Scanner scan = new Scanner(new File(“texto.txt”));
scan.useDelimiters(“”);
int cont = 0;
int estado = 0;
while (scan.hasNext()) {
   String act = scan.next();
   switch(estado) {
   case 0:
       if(act.equals(“c”)) {
           estado = 1;
       }
       break;
   case 1:
       if(act.equals(“o”)) {
           estado = 0;
           cont++;
        } else if (act.equals(“c”)) {
           estado = 1;
        } else {
           estado = 0;
        }
        break;
     }
}
```

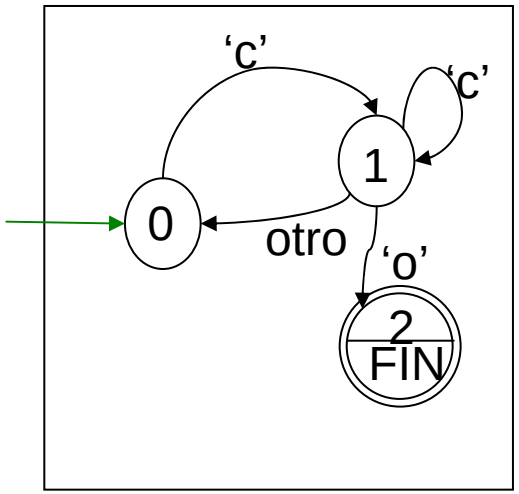
Máquina de estados (II): reconocedora

- En ocasiones no es necesario investigar toda la secuencia.
- Por ejemplo: encontrar si está la subcadena ‘co’.
- Si encontrada ‘co’, no es necesario seguir.
- Este tipo de máquinas se llaman *reconocedoras*.
- En la máquina de estados se marcan con 

  los estados que finalizan el análisis.

Máquina de estados (III): reconocedora

¿Está la subcadena "co"?



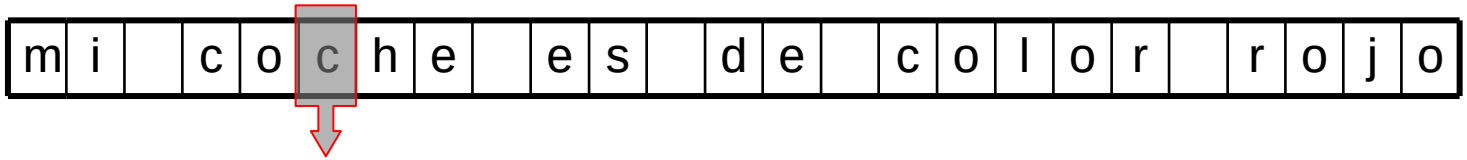

```
Scanner scan=new Scanner(new File("texto.txt"));
scan.useDelimiters("");
int cont = 0;
int estado = 0;
while (scan.hasNext() && estado != 2) {
   String act = scan.next();
   switch(estado) {
   case 0:
       if(act.equals("c")) {
           estado = 1;
       }
       break;
   case 1:
       if(act.equals("o")) {
           estado = 2;
       } else if (act.equals("c")) {
           estado = 1;
        } else {
           estado = 0;
        }
        break;
   }
}
//al salir bucle comprobar si estamos en 2
if (estado == 2) {
   System.out.println("Encontrada");
} else {
   System.out.println("No encontrada");
}
```

Máquina de estados (IV)

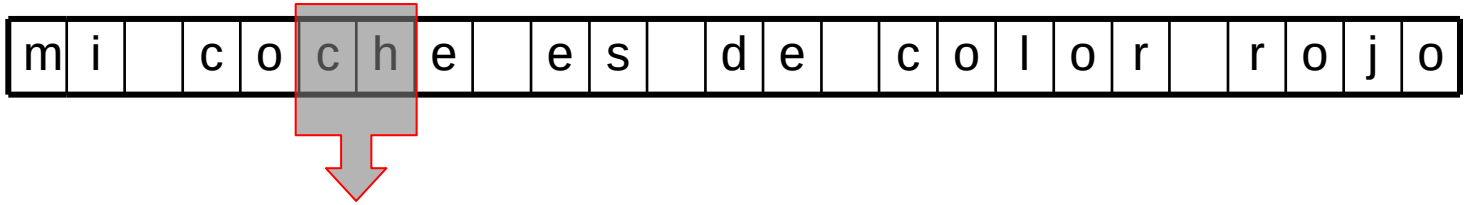
- El problema anterior se podría resolver más fácilmente
- Consideremos que la secuencia se compone de pares:

(car anterior, car actual)

Secuencia carácter a caracter

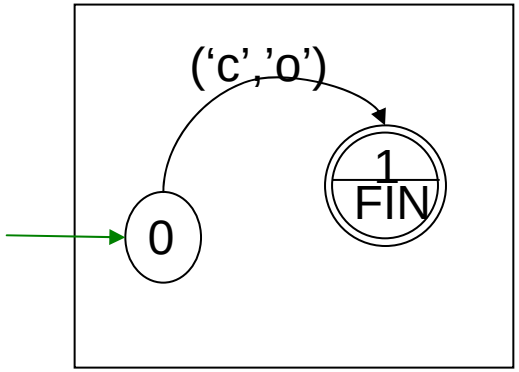


Secuencia pares de caracteres



TEMA 4 : Comunicación y algoritmos

Máquina de estados (V)



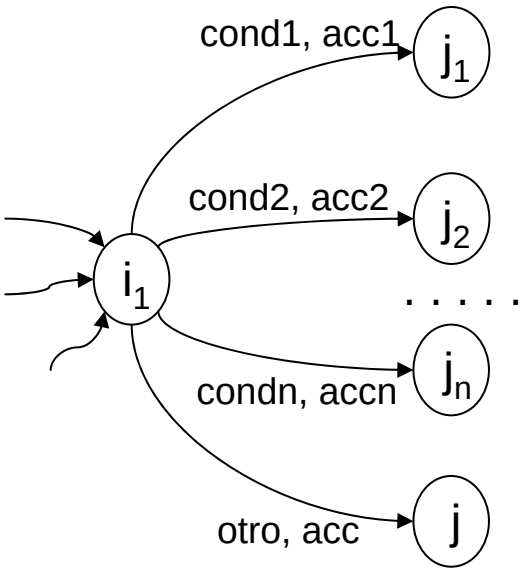

```
Scanner scan=new Scanner(new File("texto.txt"));
scan.useDelimiters("");
int cont = 0;
int estado = 0;
String ant = "";
String act = "";
while (scan.hasNext() && estado != 1) {
   ant = act;
   act = scan.next();
   switch(estado) {
   case 0:
       if(ant.equals("c") && act.equals("o")) {
           estado = 1;
       }
       break;
   }
}
//al salir bucle comprobar si estamos en 1
if (estado == 1) {
   System.out.println("Encontrada");
} else {
   System.out.println("No encontrada");
}
```

Máquina de estados: Resumen (I)

- Para resolver un problema de análisis de secuencias:
  1. Elegir bien los elementos de la secuencia.
  2. Pintar máquina de estados
  3. Escribir código

TEMA 4 : Comunicación y algoritmos

Máquina de estados: Resumen (II)




```
while (hay elem secuenc && no estados finales) {
  //siguiente elemento secuencia
  switch(estado) {
  case 0:
       . . . .
       break;
   . . . .
   case i1:
        if (cond1) {
           estado = j1;
           //hacer acciones acc1
        } else if (cond2) {
           estado = j2;
           //hacer acciones acc2
        } else . . . .
           . . . .
        } else if (condn) {
           estado = jn;
           //hacer acciones accn
        } else {
            estado = j;
            //hacer acciones acc
        }
        break;
   case . . . .
   }
}
```