

# Parte I: Programación en un lenguaje orientado a objetos

---

1. Introducción a los lenguajes de programación

2. Datos y expresiones

**3. Estructuras algorítmicas**

- Instrucción condicional. Instrucción condicional múltiple. Instrucciones de bucle. Recursión. Descripción de algoritmos mediante pseudocódigo.

4. Datos compuestos

5. Tratamiento de errores

6. Entrada/salida

7. Herencia y polimorfismo

# 3.1 Introducción

---

Las estructuras algorítmicas permiten componer instrucciones de un computador para que se ejecuten en el orden deseado

| Estructura algorítmica  | Descripción   | Instrucción  |
|-------------------------|---|--|
| Composición secuencial  | Las instrucciones se ejecutan en secuencia, una tras otra                 | Ninguna. Simplemente se ponen las instrucciones una detrás de otra |
| Composición alternativa | En función de una condición se eligen unas instrucciones u otras          | Instrucciones de control: condicionales                            |
| Estructura iterativa    | Se repiten unas instrucciones mientras se cumple una condición            | Instrucciones de control: bucles                                   |
| Estructura recursiva    | Se repiten unas instrucciones mediante un método que se invoca a sí mismo | Método o función que se invoca a sí mismo                          |

# Instrucciones simples y compuestas

---

Las instrucciones de un programa Java pueden ser:

- ***simples***:

- expresiones: de asignación, incremento o decremento
- llamadas a métodos
- creación de objetos
- instrucciones de control: `if`, `switch`, `while`, `do-while`, `for`

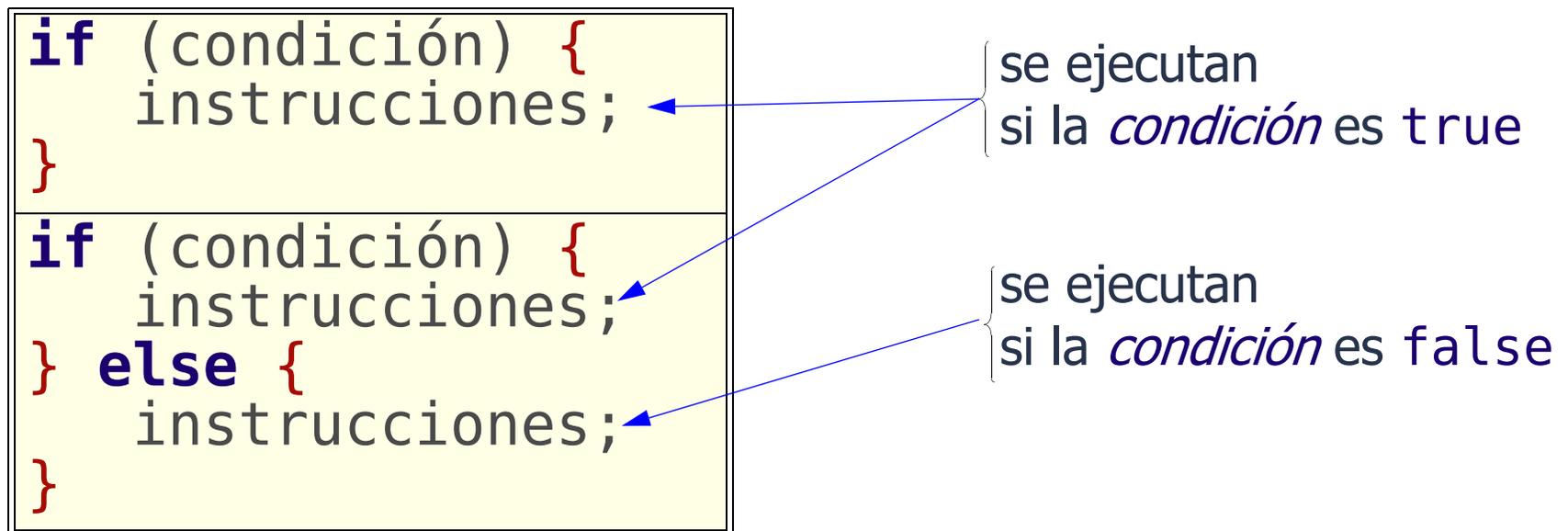
- ***compuestas***:

- pueden sustituir a una instrucción simple
- se encierran entre llaves `{}`, y también se llaman ***bloques***
- pueden contener muchas instrucciones y declaraciones
- las declaraciones del bloque sólo son visibles en él, y en los bloques contenidos en él

## 3.2. Instrucción condicional simple

---

La instrucción condicional simple permite tomar decisiones empleando una variable booleana:



La condición: expresión booleana (lógica o relacional)

# La instrucción condicional simple (cont.)

---

También se puede escribir como instrucción simple, aunque es menos recomendable (por ser menos visible el comienzo y final):

```
if (condición)  
    instrucción;  
else  
    instrucción;
```

# Ejemplo

---

Poner un texto aprobado o suspenso según la nota

```
if (nota >= 5.0) {  
    System.out.println("Aprobado");  
} else {  
    System.out.println("Suspenso");  
}
```

# Instrucciones condicionales anidadas

---

Las instrucciones `if` también se pueden anidar:

- el `else` se asocia al `if` anterior más próximo que no tenga `else`, siempre que esté en el mismo bloque que el `else`.

Ejemplo: poner "cum laude" en el ejemplo anterior si `nota >= 9`

```
if (nota >= 5.0) {
    System.out.print("Aprobado");
    if (nota >= 9.0) {
        System.out.println(" cum laude");
    } else {
        System.out.println("");
    }
} else {
    System.out.println("Suspenso");
}
```

# Expresiones condicionales

---

Como expresión condicional se pueden usar operaciones relacionales y lógicas

Ejemplo: Intervalo: condición  $a \in (5.0, 6.3]$

```
if (a > 5.0 && a <= 6.3) ...
```

Ejemplo: Intervalo contrario: condición  $a \notin (5.0, 6.3]$

```
if (a <= 5.0 || a > 6.3) ...
```

```
if (!(a > 5.0 && a <= 6.3)) ...
```

# Ejemplo: año bisiesto

```
boolean esBisiesto;
int año=...;

if (año % 4 == 0) {
    if (año % 100 == 0) {
        if (año % 400 == 0) {
            esBisiesto=true;
        } else {
            esBisiesto=false;
        }
    } else {
        esBisiesto=true;
    }
} else {
    esBisiesto=false;
}
```

Son bisiestos los años múltiplos de 4, excepto los múltiplos de 100 que no sean múltiplos de 400

# Ejemplo: año bisiesto (cont.)

---

Otra alternativa usando expresiones lógicas:

```
esBisiesto=(año % 4 == 0) &&  
            !((año % 100 == 0) &&  
              !(año % 400 == 0));
```

Mostrar en pantalla el resultado:

```
if (esBisiesto) {  
    System.out.println  
        ("El año "+año+" es bisiesto");  
} else {  
    System.out.println  
        ("El año "+año+" no es bisiesto");  
}
```

# 3.3. Instrucción condicional múltiple

---

Permite tomar una decisión de múltiples posibilidades, en función de un valor no booleano

- Si este valor es discreto (`byte`, `short`, `int`, `long`, `char`, o ***enumerado***), podemos utilizar una instrucción `switch`
- A partir de Java 7 también se pueden usar valores del tipo `String`

# Instrucción condicional múltiple (cont.)

```
switch (expresión discreta o String)
```

```
{
```

```
case valor1:  
    instrucciones;
```

```
    break;
```

```
case valor2:  
    instrucciones;
```

```
    break;
```

```
case valor3:
```

```
case valor4:  
    instrucciones;
```

```
    break;
```

```
default:
```

```
    instrucciones;
```

```
}
```

se ejecutan  
si la *expresión* es valor1

se ejecutan  
si la *expresión* es valor2

se ejecutan si la *expresión*  
es valor3 o valor4

se ejecutan si la *expresión*  
es otro valor

Observar el sangrado de los `case` y `default` alineados con el `switch`

# Instrucción switch (cont.)

---

El funcionamiento es el siguiente:

- se evalúa la expresión y se salta directamente al caso que corresponde
- se ejecutan las instrucciones puestas bajo ese caso, y todas las siguientes que se encuentren, hasta encontrar un **break**
- si no coincide con ningún caso, se ejecutan las instrucciones que haya en la parte `default`, si existe
- después de un **break**, la instrucción `switch` termina y seguimos por la siguiente instrucción
- los valores deben ser *constantes*, no variables
- no puede haber ninguno coincidente

# Ejemplo: nota media (entera) con letra

---

```
/**
 * Clase que contiene la nota media de un alumno
 */
public class NotaEntera {

    private int notaMedia;

    /**
     * Constructor al que se le pasa la nota media
     */
    public NotaEntera (int nota) {
        notaMedia=nota;
    }
}
```

# Ejemplo: nota media (entera) con letra (cont.)

---

```
/**
 * Convierte la nota media a letra
 * (sobresaliente, notable,...)
 */
public String convierte() {
    String notaLetra;

    switch (notaMedia) {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
            notaLetra="Suspenso";
            break;
        case 5:
        case 6:
            notaLetra="Aprobado";
            break;
    }
}
```

# Ejemplo: nota media (entera) con letra (cont.)

---

```
case 7:  
case 8:  
    notaLetra="Notable";  
    break;  
case 9:  
case 10:  
    notaLetra="Sobresaliente";  
    break;  
default:  
    notaLetra="Error";  
}  
return notaLetra;  
}  
}
```

# Instrucción condicional múltiple no discreta

---

Cuando la decisión se basa en un valor que no es discreto ni `String`, usamos una "*escalera*" de instrucciones `if`:

```
if (condición1) {
    instrucciones;
} else if (condición2) {
    instrucciones;
} else if (condición3) {
    instrucciones;
}
...
else {
    instrucciones;
}
```

# Instrucción condicional múltiple no discreta (cont.)

---

- Las condiciones se examinan empezando por la de arriba
- Tan pronto como una se cumple, sus instrucciones se ejecutan y la instrucción se abandona.
- Si ninguna de las condiciones es cierta se ejecuta la última parte `else`.

La instrucción `switch` es mucho más eficiente que la instrucción condicional múltiple

- en el `switch` sólo se toma una decisión
- en el `if` múltiple se evalúan muchas condiciones.

# Ejemplo: nota media (real) con letra

---

```
/**
 *
 * Clase que contiene la nota media de un alumno
 */
public class NotaReal {

    private double notaMedia;

    /**
     * Constructor al que se le pasa la nota media
     */
    public NotaReal(double nota) {
        notaMedia=nota;
    }

    /**
     * Convierte la nota media a letra (sobresaliente, notable,
     * aprobado, suspenso
     */
}
```

# Ejemplo: nota media (real) con letra (cont.)

---

```
public String convierte() {
    String notaLetra;
    if (notaMedia<0.0) {
        notaLetra="Error";
    } else if (notaMedia<5.0) {
        notaLetra="Suspenso";
    } else if (notaMedia<7.0) {
        notaLetra="Aprobado";
    } else if (notaMedia<9.0) {
        notaLetra="Notable";
    } else if (notaMedia<=10.0) {
        notaLetra="Sobresaliente";
    } else {
        notaLetra="Error";
    }
    return notaLetra;
}
}
```

## 3.4. Instrucciones de lazo o bucle

---

Permiten ejecutar múltiples veces unas instrucciones

- se corresponden a la ***composición iterativa***

La cantidad de veces se puede establecer mediante:

- ***una condición:***
  - se comprueba ***al principio***: las instrucciones del bucle se hacen cero o más veces
  - se comprueba ***al final***: las instrucciones del bucle se hacen una o más veces
- ***un número fijo de veces:*** se usa una variable de control

# 3.4.1. Bucle con condición de permanencia al principio

---

Es el bucle `while`:

```
while (condicion) {  
    instrucciones;  
}
```

se ejecutan mientras la  
condición sea `true`



Se ejecuta cero o más veces

# Ejemplo

---

Calcular el primer entero positivo tal que la suma de él y los anteriores sea mayor que 100

```
/**
 * Programa que calcula el primer entero positivo tal que la
 * suma de él y los anteriores sea mayor que 100
 */
public class SumaMayor100 {

    public static void main(String[] args) {
        int suma =0;
        int i=0;
        while (suma<=100) {
            i++;
            suma=suma+i;
        }
        System.out.println("La suma de i=1.." +i+" es "+suma);
    }
}
```

# Ejemplo 2: bucle infinito o indefinido

---

Cálculo de las distancias entre dos puntos del globo terráqueo, múltiples veces.

```
import fundamentos.*;

/**
 * Programa que calcula las distancias entre dos puntos del globo
 * terráqueo, y que pide datos de manera repetitiva
 */
public class Dist {

    public static void main(String[] args) {

        double dist; // Kilómetros
        double lon1, lat1, lon2, lat2; // grados

        // paso 1 de la lectura: crear objeto de la clase lectura
        Lectura pantalla = new Lectura("Círculo Máximo");
```

# Ejemplo 2: bucle infinito o indefinido (cont.)

---

```
// paso 2 de la lectura: crear entradas  
pantalla.creaEntrada("Latitud 1",0.0);  
pantalla.creaEntrada("Longitud 1",0.0);  
pantalla.creaEntrada("Latitud 2",0.0);  
pantalla.creaEntrada("Longitud 2",0.0);
```

# Ejemplo 2: bucle infinito o indefinido (cont.)

```
while (true) {  
    // paso 3 de la lectura: esperar  
    pantalla.espera("Introduce coordenadas y pulsa OK");  
    // paso 4 de la lectura: obtener los datos tecleados  
    lat1=pantalla.leeDouble("Latitud 1");  
    lon1 =pantalla.leeDouble("Longitud 1");  
    lat2 =pantalla.leeDouble("Latitud 2");  
    lon2 =pantalla.leeDouble("Longitud 2");  
    // cálculo de la respuesta  
    lat1=Math.toRadians(lat1);    lat2=Math.toRadians(lat2);  
    lon1=Math.toRadians(lon1);    lon2=Math.toRadians(lon2);  
    dist=Math.toDegrees(Math.acos(Math.sin(lat1)*  
        Math.sin(lat2)+  
        Math.cos(lat1)*Math.cos(lat2)*Math.cos(lon1-lon2)))*  
        60.0*1.852;  
    pantalla.println("La distancia es: "+dist+" Km");  
}
```

## 3.4.2. Bucle con condición de permanencia al final

---

Es el bucle `do-while`:

```
do {  
    instrucciones;  
} while (condicion);
```

se ejecutan mientras la  
condición sea `true`



Se ejecuta una o más veces

# Ejemplo

---

Calcular el máximo de unos números positivos introducidos por teclado, hasta que el usuario no quiera seguir

```
import fundamentos.*;

/**
 * Programa que va pidiendo datos por teclado y calcula su máximo
 */
public class Maximo {

    public static void main(String[] args) {

        double max = -Double.MAX_VALUE; // El mínimo valor posible
        double num;
        String seguir;

        Lectura l = new Lectura("Máximo");
        l.creaEntrada("Número", 0.0);
        l.creaEntrada("Seguir? (s/n)", "s");
```

# Ejemplo (cont.)

---

```
do {  
    l.espera("Introd. número y pulsa OK");  
    num = l.leeDouble("Número");  
    seguir = l.leeString("Seguir? (s/n)");  
    if (num>max) {  
        max=num;  
    }  
    l.println("El máximo es: "+max);  
} while (seguir.equalsIgnoreCase("s"));  
  
l.println("Pulsa Cerrar");  
  
}
```

## 3.4.3 Bucle con variable de control

---

Es el bucle `for`:

```
for (decl-inicialización; cond-permanencia; expr-incremento)
{
    instrucciones;
}
```

Es equivalente a:

```
{
    decl-inicialización;
    while (cond-permanencia) {
        instrucciones;
        expr-incremento;
    }
}
```

# Ejemplos

---

Suma de los 100 primeros enteros positivos:

```
int suma=0;
for (int i=1; i<=100; i++) {
    suma=suma+i;
}
```

También para incrementos distintos de uno (ej: nº pares):

```
int suma=0;
for (int i=2; i<=100; i=i+2) {
    suma=suma+i;
}
```

# Recomendaciones sobre el bucle for

---

- Debe usarse para bucles con variable de control y de una manera uniforme
- Es conveniente declarar la variable de control en el bucle
- Es conveniente que la expresión de incremento sea eso
- Es conveniente que la expresión de permanencia sea sencilla
- *Nunca cambiar el valor* de la variable de control en las instrucciones

# Variantes de bucles

---

Hacia atrás:

```
for (int n=10; n>=-6; n--) ...
```

Vacío:

```
for (int n=0; n<finish; n++) ...//si finish<0
```

Anidado

```
for (int i=1; i<=10; i++) {  
    for (int j=1; j<=20; j++) {  
        ...  
    }  
}
```

# Ejemplo: uso de la clase Grafica

---

Es una clase sencilla para hacer gráficos de funciones reales. Permite:

- almacenar puntos
- mostrarlos como puntos o líneas
- mostrar el gráfico
- puede mostrar varios gráficos en la misma ventana

# Ejemplo (cont.)

---

```
import fundamentos.*;
/** Programa que muestra gráficas del seno y el coseno */

public class FuncionesTrigonometricas {

    public static void main(String[] args) {

        // Gráficas de funciones trigonometricas
        Grafica g = new Grafica ("Seno y Coseno", "x", "y");
        double x;

        // El primer gráfico
        g.ponSimbolo(true);
        g.ponColor(Grafica.azul);
        g.ponTitulo("Seno");
        // Ángulos desde 0 a 3*PI con incremento de PI/16
        for (double x1=0.0; x1<=Math.PI*3.0; x1=x1+Math.PI/16.0)
        {
            g.inserta(x1,Math.sin(x1));
        }
    }
}
```

# Ejemplo (cont.)

---

```
// El segundo gráfico
g.otraGrafica();
g.ponSimbolo(true);
g.ponColor(Grafica.rojo);
g.ponTitulo("Coseno");
// Ángulos desde 0 a 10 radianes con incremento
// de 0.1 radianes
for (int i=0; i<=100; i++) {
    x = i/10.0;
    g.inserta(x,Math.cos(x));
}

// Pinta ambos gráficos
g.pinta();
}
```

## 3.4.4. Instrucciones de salto en bucles

---

Hay tres instrucciones que permiten saltarse las instrucciones restantes del bucle:

- **break**:
  - termina el bucle
- **continue**:
  - termina las instrucciones del bucle, pero sigue en él
- **return**:
  - termina un método; si estamos en un bucle, lógicamente también lo termina

Se recomienda no usarlas

- las que terminan el bucle se pondrían con condiciones adicionales de permanencia en el bucle
- el **continue** se resuelve con una instrucción condicional

# 3.5. Recursión

---

Muchos algoritmos iterativos pueden resolverse también con un algoritmo ***recursivo***

- el algoritmo se invoca a sí mismo
- en ocasiones es más natural
- en otras ocasiones es más engorroso

El ***diseño recursivo*** consiste en diseñar el algoritmo mediante una estructura condicional de dos ramas

- ***caso directo***: resuelve los casos sencillos
- ***caso recursivo***: contiene alguna llamada al propio algoritmo que estamos diseñando

# Ejemplos

---

Definición iterativa para el cálculo del **factorial** de un número natural

$$n! = 1, n = 0$$

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n, n \geq 1$$

Definición recursiva para el cálculo del **factorial** de un número natural

$$n! = 1, n = 0$$

$$n! = n \cdot (n-1)!, n \geq 1$$

La definición es correcta pues el número de recursiones es finito

# Ejemplo: factorial recursivo

---

```
/**
 * Retorna el factorial de n
 */
int factorial (int n) {
    if (n<=1) {
        // caso directo
        return 1;
    } else {
        // caso recursivo
        return n*factorial(n-1);
    }
}
```

# Fases del diseño recursivo

---

Obtener una definición recursiva de la función a implementar a partir de la especificación

- Establecer caso directo
- Establecer caso recursivo

Diseñar el algoritmo con una instrucción condicional

Argumentar sobre la terminación del algoritmo

# Ejemplo 2: Convertir un número decimal a otra base de numeración

---

Variables:

- $x$ : número entero a convertir
- $b$ : base destino ( $2 \leq b \leq 10$ )
- El resultado se muestra en pantalla

Caso directo

- si  $x < b$ , el resultado es  $x$

Caso recursivo

- convertir  $x/b$  a la base  $b$  (invocando el mismo método)
  - se trabaja con la parte más significativa
- y luego mostrar en pantalla  $x \% b$ 
  - se trabaja con la parte menos significativa al final

# Ejemplo 2 (cont.)

---

```
/**
 * Muestra en pantalla la conversión de x a la base
 * de numeración b;
 * b debe estar entre 2 y 10
 */
public static void convertir(int x, int b) {
    if (x < b) {
        // caso directo
        System.out.print(x);
    } else {
        // caso recursivo
        convertir(x/b, b);
        System.out.print(x%b);
    }
}
```

# Consideraciones sobre los datos

---

Datos compartidos por todas las invocaciones del algoritmo

- atributos del objeto
- estado de otros objetos externos

Datos para los que cada invocación tiene una copia posiblemente distinta

- variables locales (internas) del algoritmo
- parámetros
- valor de retorno del método

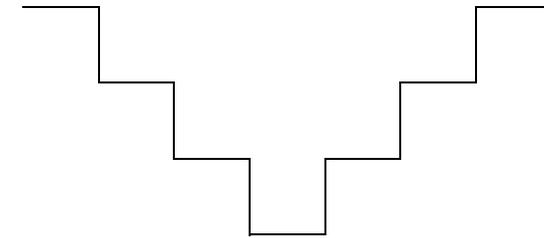
# Ejemplo: pintar un valle de n escalones

Proceso:

- caso directo ( $n=1$ )
  - pintar el fondo del valle
- caso recursivo ( $n>1$ )
  - pintar un escalón de bajada
  - pintar un valle de  $n-1$  escalones
  - pintar un escalón de subida

Al finalizar cada paso hay que dejar el lápiz en la posición correcta para el paso siguiente

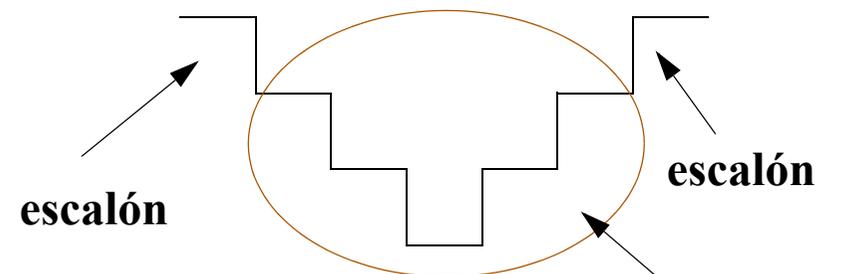
- el lápiz es un objeto externo, compartido por todos los pasos



a) valle de 4 escalones



b) caso directo



c) caso recursivo

# Ejemplo (cont.)

---

Para este ejemplo disponemos de una clase para pintar líneas

Disponemos asimismo de un atributo de esta clase llamado  $\ell$ :

Lapiz  $\ell$ ;

- `mueveLapiz()` mueve el lápiz en la dirección actual la distancia indicada
  - la dirección inicial es hacia la derecha
- `gira()` gira el lápiz el ángulo indicado en sentido antihorario

| Lapiz  |
|--|
| ...  |
| +Lapiz()<br>+void mueveLapiz(double distancia)<br>+void gira(double angulo)<br>... |

# Ejemplo (cont.)

---

```
/**
 * Metodo que pinta un valle de n escalones
 */
public void valle(int n) {
    if (n==1) {
        // en el caso directo pinta el fondo del valle
        l.mueveLapiz(10.0);
    } else {
        // en el caso recursivo se pinta un escalón
        // de bajada, un valle de n-1 escalones,
        // y un escalón de subida

        // escalón de bajada
        l.mueveLapiz(10.0);
        l.gira(-90.0);
        l.mueveLapiz(10.0);
        l.gira(90.0);
    }
}
```

# Ejemplo (cont.)

---

```
// valle de n-1 escalones  
// (se llama recursivamente al mismo método)  
valle(n-1);
```

```
// escalón de subida  
l.gira(90.0);  
l.mueveLapiz(10.0);  
l.gira(-90.0);  
l.mueveLapiz(10.0);
```

```
}  
}
```

# 3.6. Descripción de algoritmos mediante pseudocódigo

---

Una técnica muy habitual para describir algoritmos es el pseudocódigo. Tiene como objetivos:

- descripción *sencilla*, sin los formalismos de un lenguaje de programación
- descripción *independiente del lenguaje* de programación
  - directamente traducible a código en cualquier lenguaje

El pseudocódigo contiene:

- instrucciones de control presentes en todos los lenguajes
- declaraciones de datos
- expresiones con cálculos
- acciones expresadas sin el formalismo de los lenguajes

# Pseudocódigo: Instrucciones de control

| condicional   | switch  |
|---|---|
| <pre><b>si</b> condición <b>entonces</b>   instrucciones <b>si no</b>   instrucciones <b>fin si</b></pre> | <pre><b>si</b> valor   <b>caso</b> a=&gt; instrucciones   <b>caso</b> b=&gt; instrucciones   <b>ninguno de los anteriores</b>     =&gt; instrucciones <b>fin si</b></pre> |
| bucle while   | bucle do-while  |
| <pre><b>mientras</b> condición   instrucciones <b>fin mientras</b></pre>                                  | <pre><b>hacer</b>   instrucciones <b>mientras</b> condición</pre>   |
| bucle for   | bucle for que recorre una lista   |
| <pre><b>para</b> i <b>desde</b> 1 <b>hasta</b> n   instrucciones <b>fin para</b></pre>                    | <pre><b>para</b> cada x <b>en</b> lista   instrucciones <b>fin para</b></pre>   |

- en el bucle *para* los valores inicial y final están *incluidos*

# Pseudocódigo: Datos, acciones y expresiones

---

Declaraciones de variables; ejemplos:

```
entero i
real temperatura
String s
// array de reales
real[0..n-1] a // el tamaño del array sería n
```

Expresiones con cálculos; ejemplo:

```
i=suma+3*x
```

Acciones expresadas sin el formalismo de los lenguajes; ejemplos:

```
leer i y j de teclado
escribir resultado en la pantalla
```

Invocar un método se hace como en Java:

```
objeto.metodo(datos)
```

# Pseudocódigo: Métodos

---

Usaremos esta estructura para definir un método que no retorna nada (`void`):

```
método nombre (parámetros)  
    instrucciones  
fin método
```

Para un método que retorna un valor:

```
método nombre (parámetros) retorna tipoRetornado  
    instrucciones  
fin método
```

# Ejemplo: suma de los 100 primeros enteros positivos

| Java  | Pseudocódigo   |
|---|--|
| <pre>int suma=0; for (int i=1; i&lt;=100; i++) {     suma=suma+i; }</pre> | <pre>entero suma=0; para i desde 1 hasta 100     suma=suma+i; fin para</pre> |

También para incrementos distintos de uno (ej: n<sup>o</sup> pares):

| Java  | Pseudocódigo  |
|---|---|
| <pre>int suma=0; for (int i=2; i&lt;=100; i=i+2) {     suma=suma+i; }</pre> | <pre>entero suma=0; para i desde 2 hasta 100 paso 2     suma=suma+i; fin para</pre> |

# Ejemplo: recorrido de un array de reales

Recorriendo todos los índices

| Java  | Pseudocódigo  |
|---|---|
| <pre>int n=12; double[] a= new double[n]; <b>for</b> (int i=0; i&lt;n; i++) {     System.out.println(a[i]); }</pre> | <pre>entero n=12; real[0..n-1] a <b>para</b> i <b>desde</b> 0 <b>hasta</b> n-1     Mostrar a[i] en pantalla <b>fin para</b></pre> |

Recorriendo cada casilla (bucle "para cada"):

| Java   | Pseudocódigo  |
|--|---|
| <pre>int n=12; double[] a= new double[n]; <b>for</b> (double x:a) {     System.out.println(x); }</pre> | <pre>entero n=12; real[0..n-1] a <b>para cada</b> x <b>en</b> a     Mostrar x en pantalla <b>fin para</b></pre> |

# Ejemplo

---

Vamos a escribir un método para obtener el valor del *logaritmo* de  $y=1+x$  de acuerdo con el siguiente desarrollo en serie

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots + (-1)^{n-1} \frac{x^n}{n}, \quad -1 < x \leq 1$$

Para calcular de manera eficiente el signo y el numerador:

- no usaremos potencias
- el signo va cambiando de un término al siguiente
- el numerador siempre es el del término anterior por  $x$

# Diseño

---

```
método logaritmo (real y, entero n) retorna real
  real x=y-1;
  real log=0; // para recoger el resultado
  real numerador=x; // primer numerador
  entero signo=1; // primer signo
  para i desde 1 hasta n
    log=log+signo*numerador/i;
    // calculamos el numerador y el signo
    // para la próxima vez
    numerador=numerador*x;
    signo=-signo;
  fin para
  retorna log;
fin método
```