

Instrucciones condicionales

Objetivos

- 1) Diagramas de flujos
- 2) Bloques de código
- 3) Instrucción if
- 4) Instrucción if-else
- 5) Instrucción if-else-if
- 6) Anidamiento de instrucciones condicionales
- 7) Instrucción de selección
- 8) Contadores, señales y acumuladores
- 9) Ámbito de validez de una variable

Instrucciones de bucle

Objetivos

- 1) Bucle while
- 2) Bucle do-while
- 3) Técnica para obtener la salida como una cadena
- 4) Bucle for
- 5) Saltos
- 6) Ciclos
- 7) Anidamiento de bucles
- 8) Etiquetas y salto

Funciones y procedimientos

Objetivos

- 1) Características de un método: Firma y parámetros
- 2) Funciones
- 3) Procedimientos
- 4) Métodos recursivos
- 5) Menús

Instrucciones de error

Objetivos

- 1) Instrucciones de error
- 2) Crear un objeto de excepción
- 3) Instrucción throw
- 4) Instrucción try-catch

Librerías de usuario

Objetivos

- 1) Saber qué es una librería de usuario
- 2) Definir funciones y procedimientos en una librería
- 3) Uso de métodos de una librería en una aplicación
- 4) Sobrecarga de métodos

La clase String

Objetivos

- 1) Saber definir expresiones regulares. Método matches
- 2) Validar datos de entrada de tipo cadena
- 3) La clase String
- 4) La clase StringBuffer

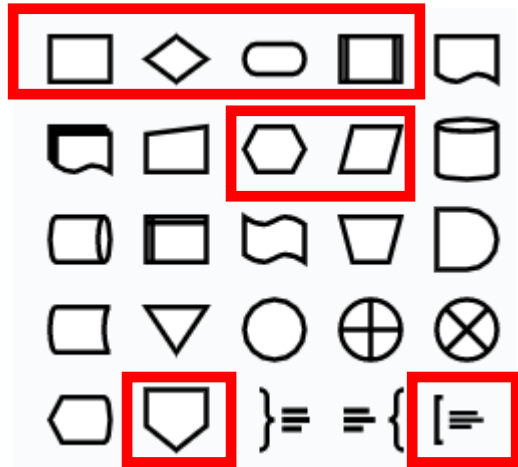
Instrucciones condicionales

Objetivos

- 1) Diagramas de flujos
- 2) Bloques de código
- 3) Instrucción if
- 4) Instrucción if-else
- 5) Instrucción if-else-if
- 6) Anidamiento de instrucciones condicionales
- 7) Instrucción de selección
- 8) Contadores, señales y acumuladores
- 9) Ámbito de validez de una variable

Diagramas de flujo

El diagrama de flujo o flujograma o diagrama de actividades es la representación gráfica de un algoritmo o proceso.



Proceso	Decisión	Inicio/Fin	Método	Documento
Documentos	Dato manual	Repetición	Entrada/salida	Base de dato
Disco duro	Almacenamiento Interno	Cinta	Operación manual	Retraso
Fichero	Mezcla	Conector	O	Unión Suma
Display	Enlace	Notas	Notas	Notas

Ejemplos de diagrama de flujo

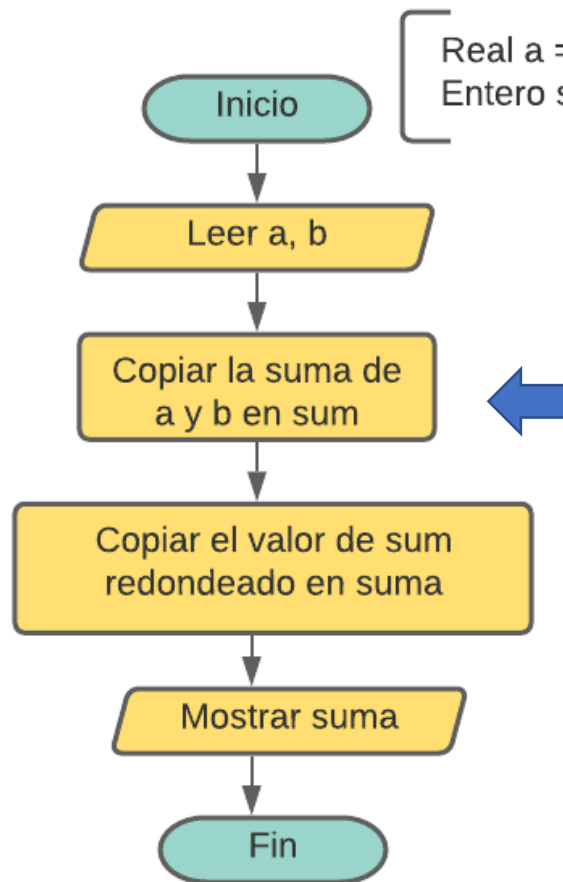


Diagrama de flujo que lee dos reales y obtiene la suma entera redondeada a las unidades

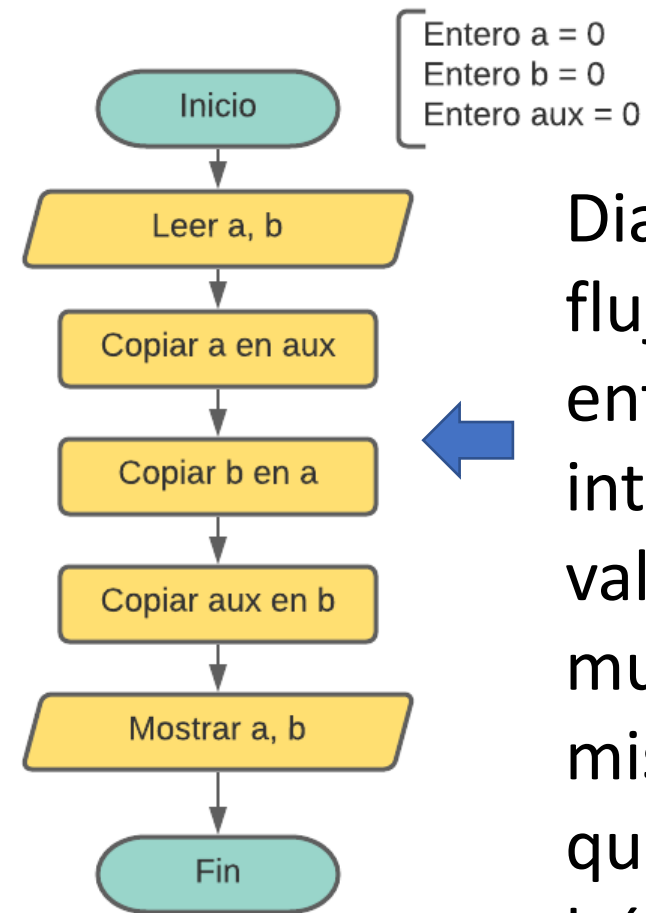


Diagrama de flujo que lee dos enteros, intercambia los valores y los muestra en el mismo orden que fueron leídos

Bloque de código

Un **bloque de código** es una sola instrucción o dos o más instrucciones entre llaves.

```
//Una instrucción  
n = n + 4;
```

```
//Dos o más instrucciones  
entre llaves  
{  
    valor = in.leerInt();  
    valor+=2;  
}
```

Instrucción if

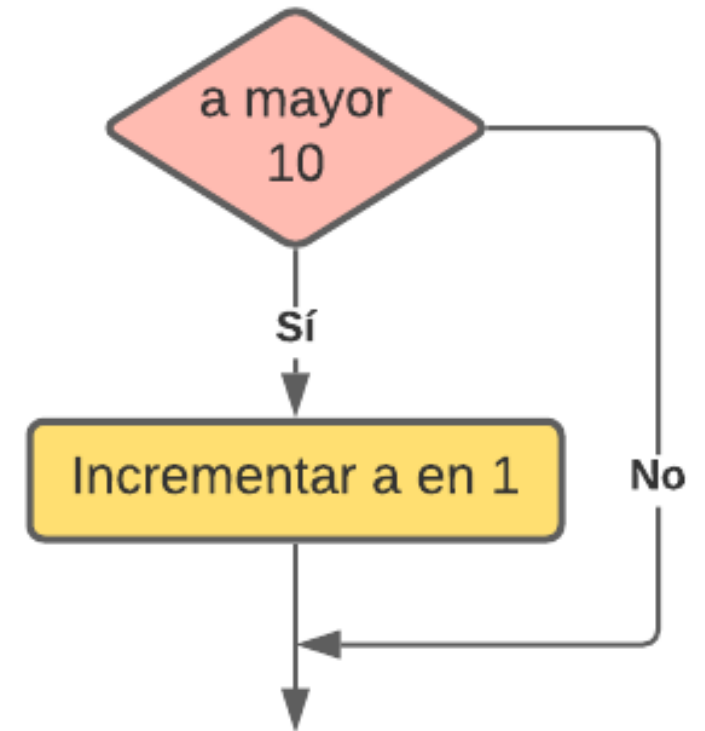
Si la expresión devuelve verdadero, entonces se ejecuta el bloque

```
if (expresion) bloque;
```

```
if (a>10) a++;
```

Algoritmo:

1. Si “a” es mayor que 10
 1. Incrementar “a” en una unidad
2. ...

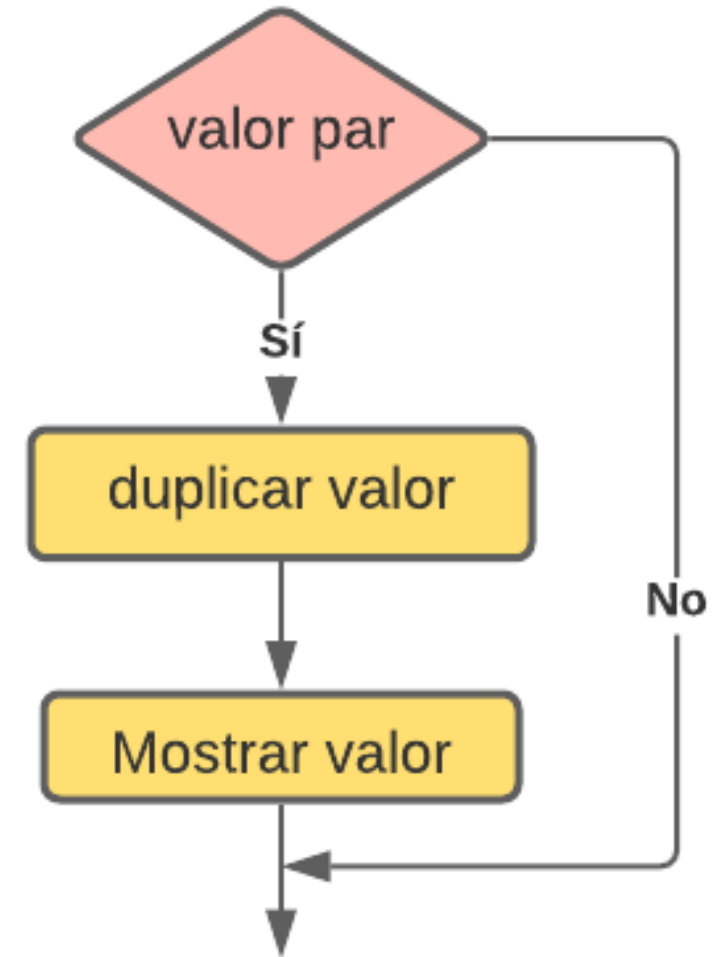


Instrucción if (II)

```
if (valor % 2 == 0)
{
    valor *= 2;
    System.out.println(valor);
}
```

Algoritmo:

1. Si “valor” es par
 1. Duplicar “valor”
 2. Mostrar “valor”
2. ...



Instrucción if-else

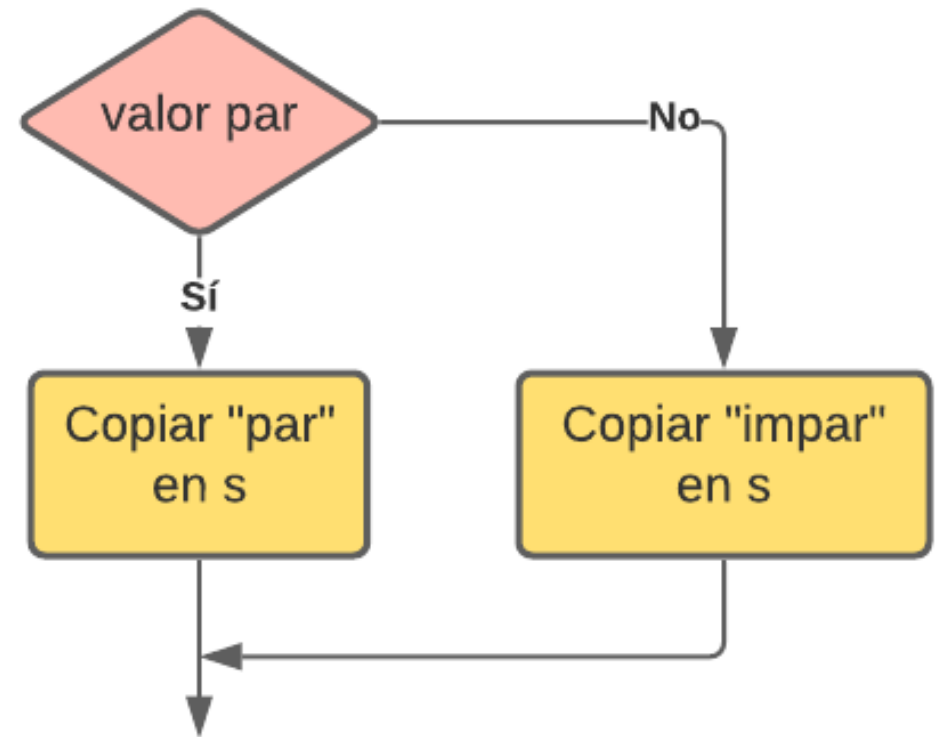
Si la expresión devuelve verdadero, entonces se ejecuta el bloque1; si no, el bloque2

```
if (expresion) bloque1;  
else bloque2;
```

```
if (valor % 2 == 0) s = "par";  
else s = "impar";
```

Algoritmo:

1. Si "valor" es par
 1. Copiar "par" en s
2. Si no
 1. Copiar "impar" en s

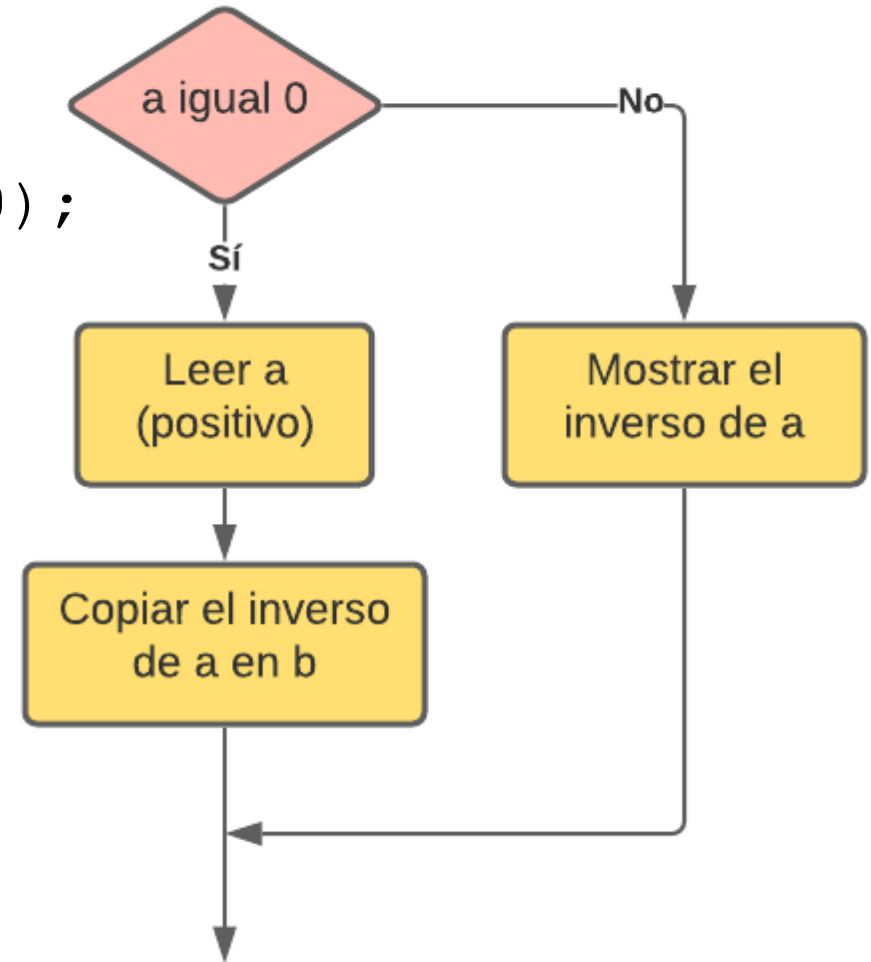


Instrucción if-else (II)

```
if (a == 0)
{
    a = in.leerDouble("REAL: ", v->v>0);
    b = 1/a;
}
else System.out.println(1/a);
```

Algoritmo:

1. Si "a" es igual a 0
 1. Leer "a" (positivo)
 2. Copiar el inverso de "a" en b
2. Si no
 1. Mostrar el inverso de "a"

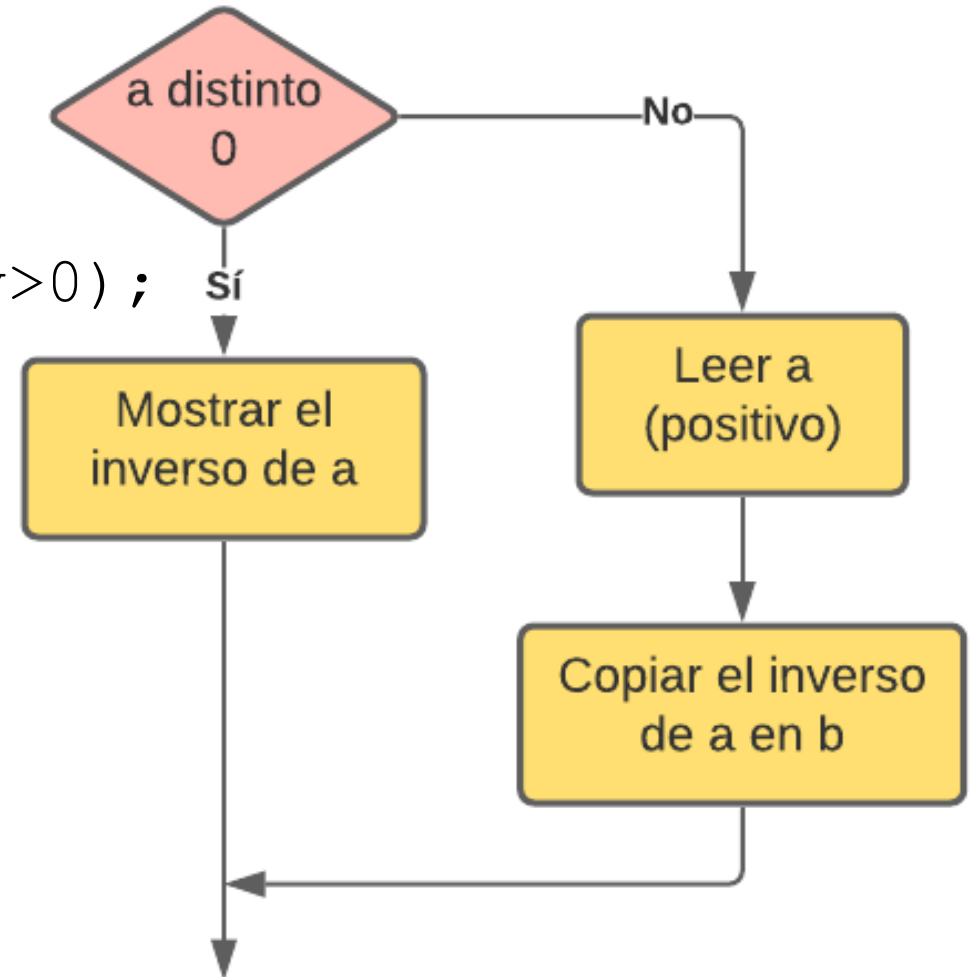


Instrucción if-else (III)

```
if (a != 0) System.out.println(a/0);  
else  
{  
    a = in.leerDouble("REAL: ", v -> v > 0);  
    b = 1/a;  
}
```

Algoritmo:

1. Si "a" es distinto de 0
 1. Mostrar el inverso de "a"
2. Si no
 1. Leer "a" (positivo)
 2. Copiar el inverso de "a" en b



Instrucción if-else-if

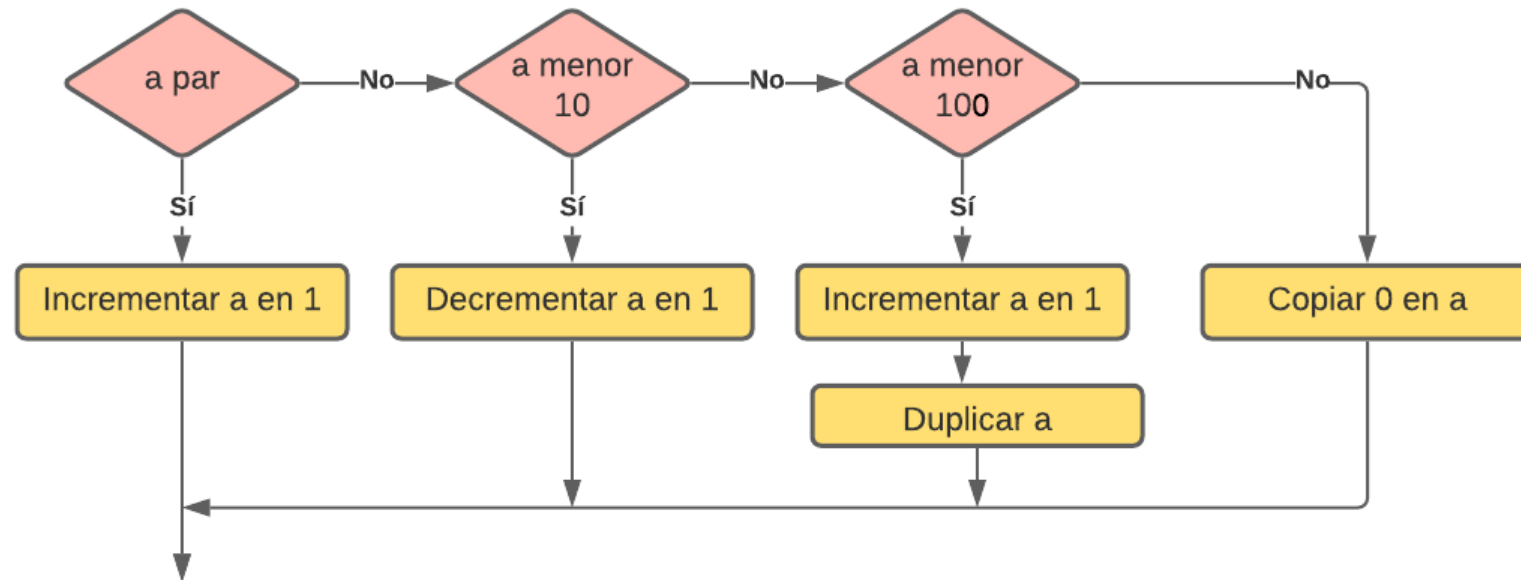
```
if(expresion1) bloque1;  
else if (expresion2) bloque2;  
...  
else bloqueN;
```

Para la primera expresión que sea cierta, se ejecuta su bloque; si no, se ejecuta el bloque del "else"

```
if (a%2==0) a++;  
else if (a<10) a--;  
else if (a<100)  
{  
    a++;  
    a*=2;  
}  
else a = 0;
```

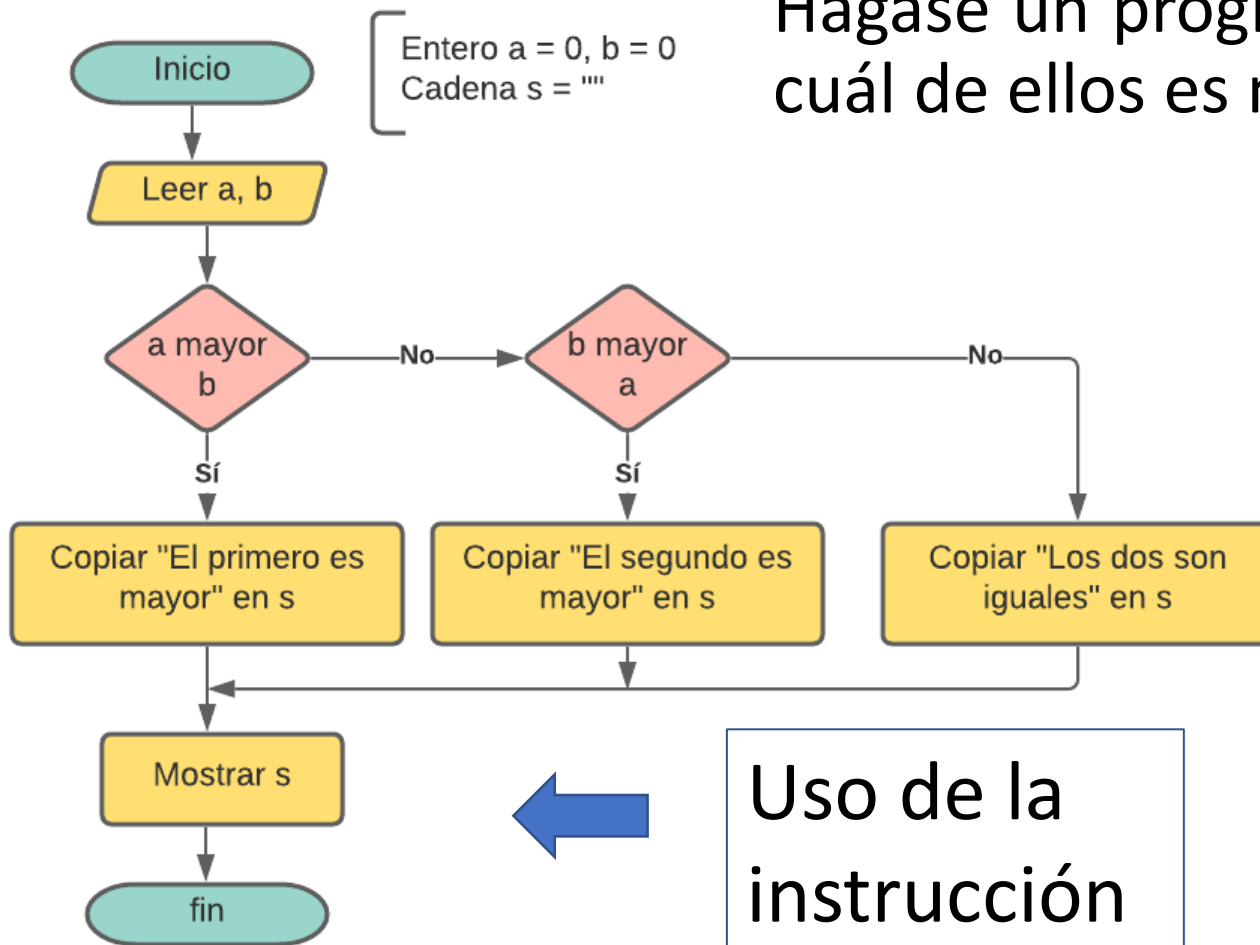
Algoritmo:

1. Si "a" es par
 1. Incrementar "a" en 1
2. Si no si "a" es menor que 10
 1. Decrementar "a" en 1
3. Si no si "a" es menor que 100
 1. Incrementar "a" en 1
 2. Duplicar "a"
4. Si no
 1. Copiar 0 en "a"



Ejemplos de instrucciones condicionales

Hágase un programa que lea dos enteros y decida cuál de ellos es mayor



Uso de la
instrucción
if-else-if

//Variables

```
int a = 0, b = 0;  
String s = "";
```

//Programa

```
a = in.leerInt();  
b = in.leerInt();
```

```
if(a>b) s = "El primero es mayor";  
else if(b>a) s = "El segundo es mayor";  
else s = "Los dos son iguales";
```

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

Tabla de ejecución

Una tabla de ejecución consiste en obtener la salida por consola de un programa para unos datos de entrada determinados y mostrar los valores que van tomando las variables y las condiciones a medida que se ejecutan las instrucciones.

Hágase una tabla de ejecución del programa anterior introduciendo como datos el 27 y 31

Instrucción/Condición	Consola/Condición	a	b	s
Inicializacion		0	0	""
a = in.leerInt()	ENTERO: 27	27	0	""
b = in.leerInt()	ENTERO: 31	27	31	""
a>b	false	27	31	""
b>a	true	27	31	""
s = "El segundo es mayor"		27	31	El segundo es mayor
System.out.println(s)	El segundo es mayor	27	31	El segundo es mayor

Ejemplos de instrucciones condicionales(II)

Hágase un programa que lea dos enteros y decida cuál de ellos es mayor

```
//Variables
```

```
int a = 0, b = 0;  
String s = "";
```

```
//Programa
```

```
a = in.leerInt();  
b = in.leerInt();  
if(a>b)  
    s = "El primero es mayor";  
if(b>a)  
    s = "El segundo es mayor";  
if(a==b)  
    s = "Los dos son iguales";  
System.out.println(s);
```

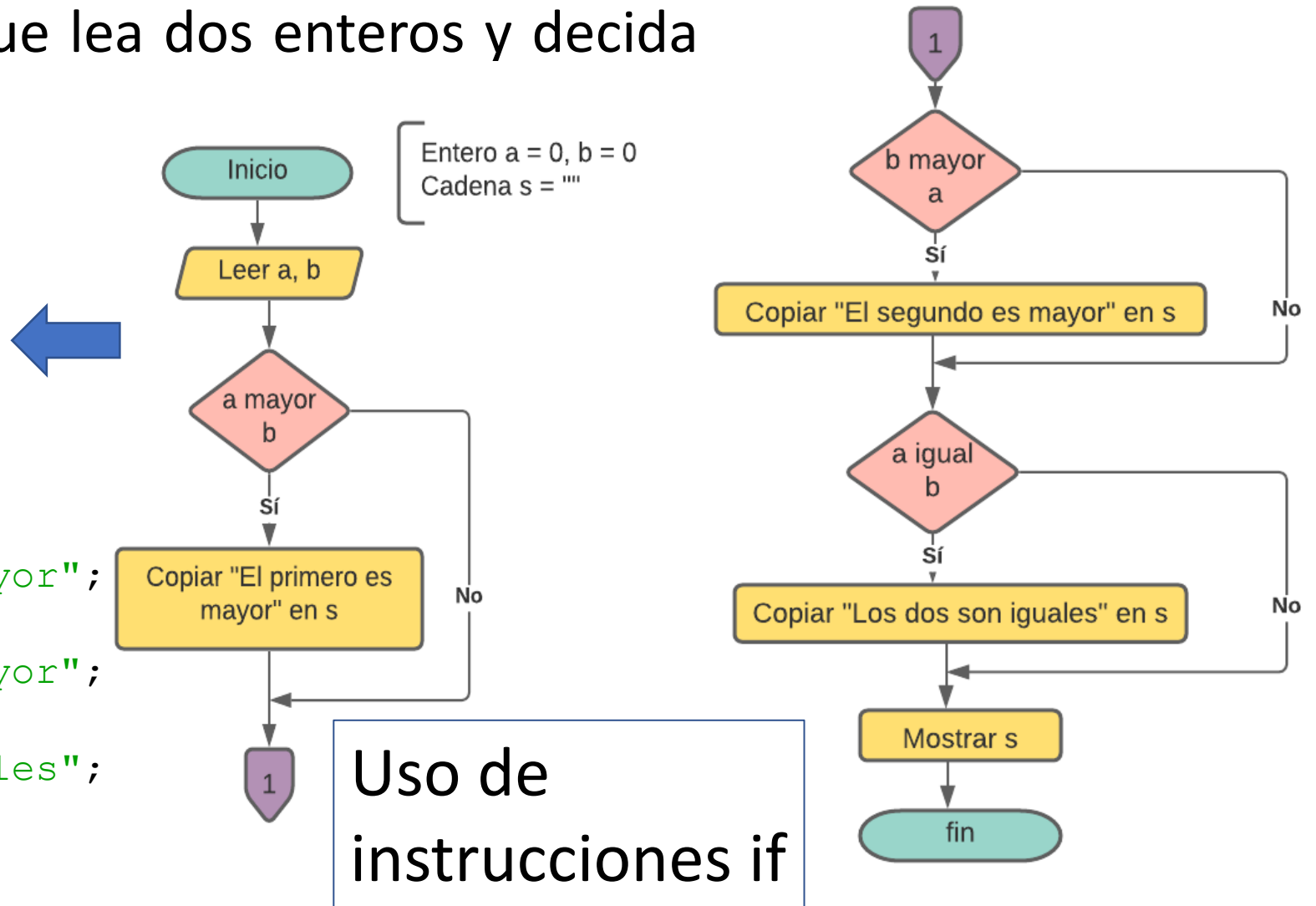


Tabla de ejecución if

Hágase una tabla de ejecución del programa anterior introduciendo como datos el 27 y 31

Instrucción/Condición	Consola/Condición	a	b	s
Inicializacion		0	0	""
a = in.leerInt()	ENTERO: 27	27	0	""
b = in.leerInt()	ENTERO: 31	27	31	""
a>b	false	27	31	""
b>a	true	27	31	""
s = "El segundo es mayor"		27	31	El segundo es mayor
b==a	false	27	31	El segundo es mayor
System.out.println(s)	El segundo es mayor	27	31	El segundo es mayor



Ejemplos de Instrucciones condicionales (III)

Hágase un programa que lea dos enteros y decida cuál de ellos es mayor

//Variables

```
int a = 0, b = 0;  
String s = "";
```

//Programa

```
a = in.leerInt();  
b = in.leerInt();  
if(a>b) s = "El primero es mayor";  
else  
{  
    s = "Los dos son iguales";  
    if(b>a)  
        s = "El segundo es mayor";  
}  
System.out.println(s);
```

Anidamiento
de if

Se hace una
apuesta

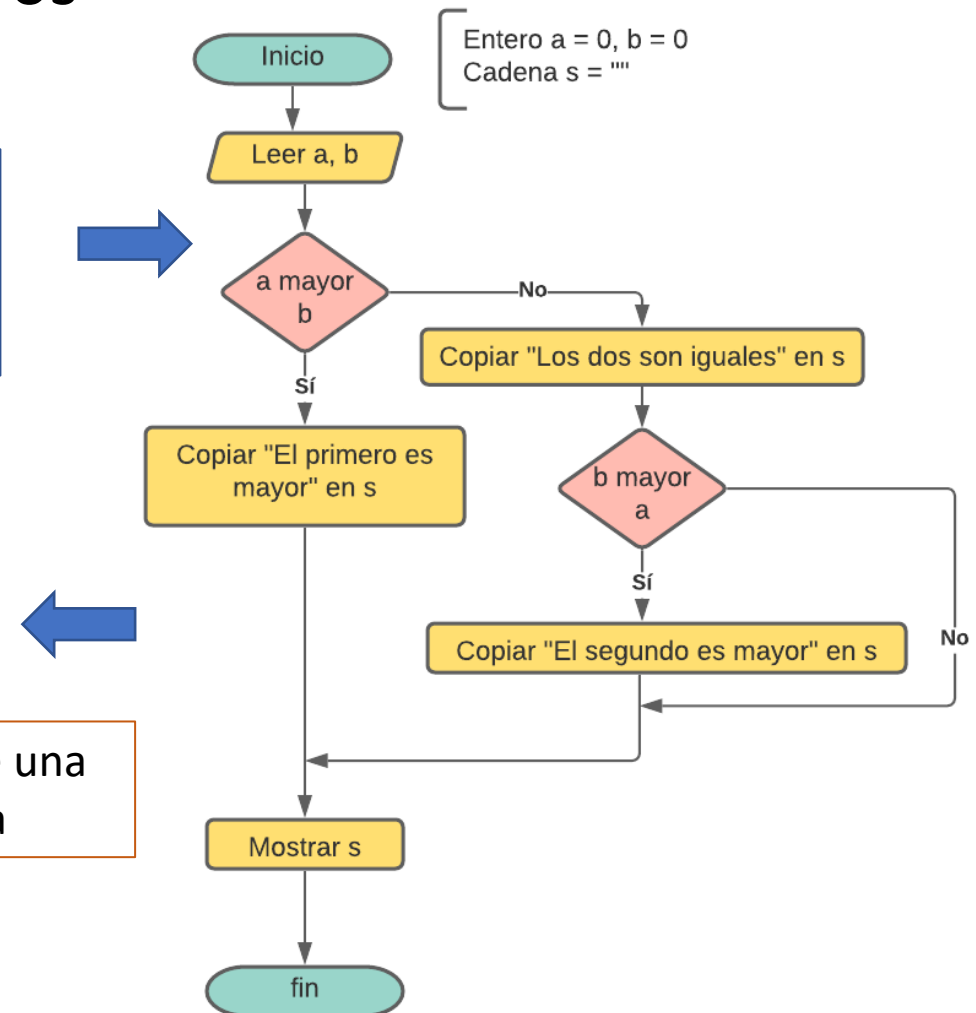


Tabla de ejecución if (II)

Hágase una tabla de ejecución del programa anterior introduciendo como datos el 27 y 31

Instrucción/Condición	Consola/Condición	a	b	s
Inicializacion		0	0	""
a = in.leerInt()	ENTERO: 27	27	0	""
b = in.leerInt()	ENTERO: 31	27	31	""
a>b	false	27	31	""
s = "Los dos son iguales"		27	31	Los dos son iguales
b>a	true	27	31	""
s = "El segundo es mayor"		27	31	El segundo es mayor
b==a	false	27	31	El segundo es mayor
System.out.println(s)	El segundo es mayor	27	31	El segundo es mayor

Instrucción de selección

`switch` (expresion)

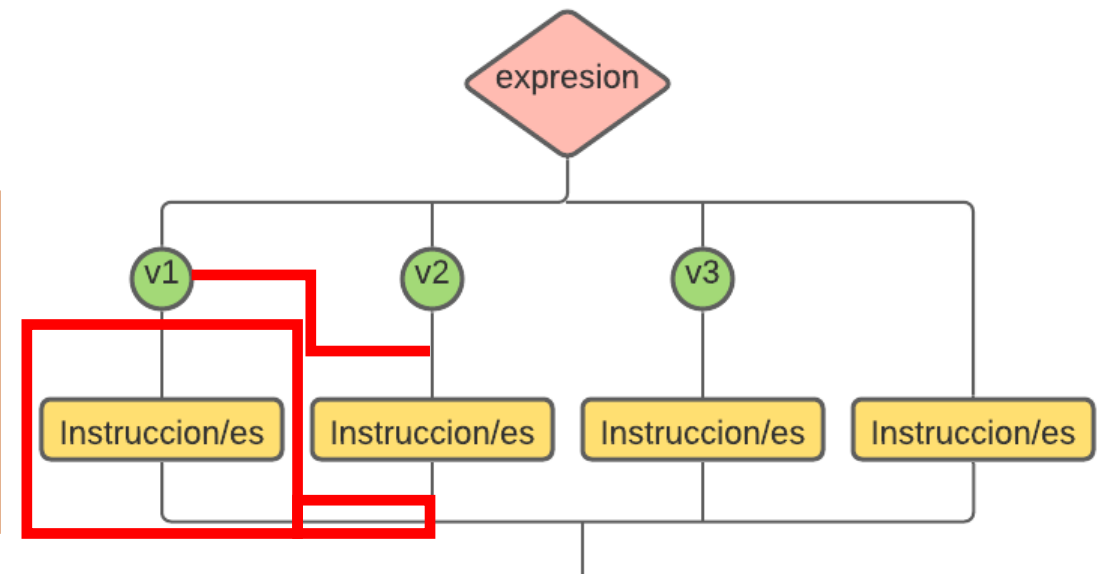
```
{  
  case v1: instruccion/es ;  
          break;  
  case v2: instruccion/es;  
          break;  
  case v3: instruccion/es;  
          break;  
  default: instruccion/es;  
}
```

La expresión tiene que devolver un valor entero (byte, short, long, int), un carácter o una cadena

Si se quitan tales instrucciones, se agrupan los valores v1 y v2

Los valores de v1, v2 y v3 pueden ser enteros (1,10, etc), caracteres ('a', 'e', etc) o cadenas ("Enero", "Febrero", etc)

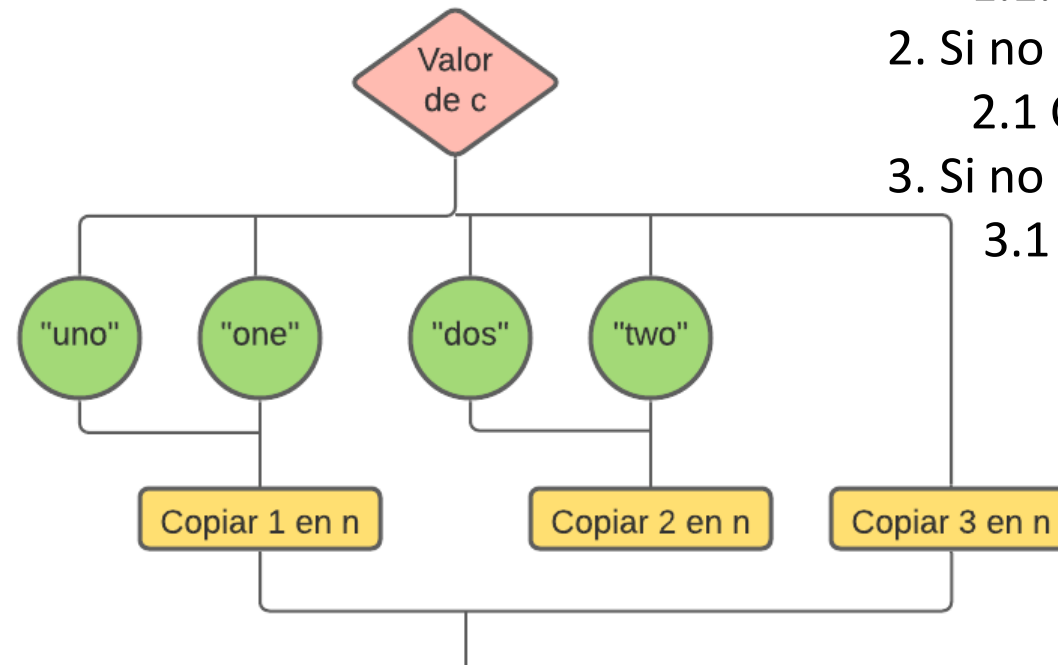
Se ejecután las instrucciones del primer valor (v1,v2, v3) coincidente con el valor de la expresión hasta el "break" (opcional) o '}' ; si no hay ningún valor coincidente, se ejecután las instrucciones del "default" (opcional)



Ejemplo de instrucción de selección

Traducir una cadena que almacena “uno”, “one”, “dos”, “two”, “tres” o “three” en el valor entero que representa

```
// "c" String
// "n" int
switch (c)
{
    case "uno":;
    case "one": n = 1;
                break;
    case "dos":;
    case "two": n = 2;
                break;
    default: n = 3;
}
```



Algoritmo

1. Si “c” es igual a “uno” o “one”
 - 1.1. Copiar 1 en “n”
2. Si no si “c” es igual a “dos” o “two”
 - 2.1 Copiar 2 en “n”
3. Si no
 - 3.1 Copiar 3 en “n”

Tabla de ejecución seleccion

Hágase una tabla de ejecución de la instrucción anterior a la que se le añade la **declaración e inicialización** de las variables “c” y “n” a “” y 0, respectivamente; la **lectura** de “c” (se introduce “dos”) y la **salida** de “n”

Instrucción/Condición	Consola/Condición	c	n
Inicializacion		""	0
c = in.leeString()	CADENA: dos	dos	0
c=="uno" c=="one"	false	dos	0
c=="dos" c=="two"	true	dos	0
n = 2		dos	2
System.out.println(n)	2	dos	2



Contadores

Los contadores son variables inicializadas a 0 y que se utilizan para realizar un recuento.

Programa que lee tres enteros y cuenta cuántos de ellos son mayores que 10

```
//Variables
```

```
int a = 0, con = 0;
```

```
//Programa
```

```
a = in.leerInt();
```

```
if(a>10) con++;
```

```
a = in.leerInt();
```

```
if(a>10) con++;
```

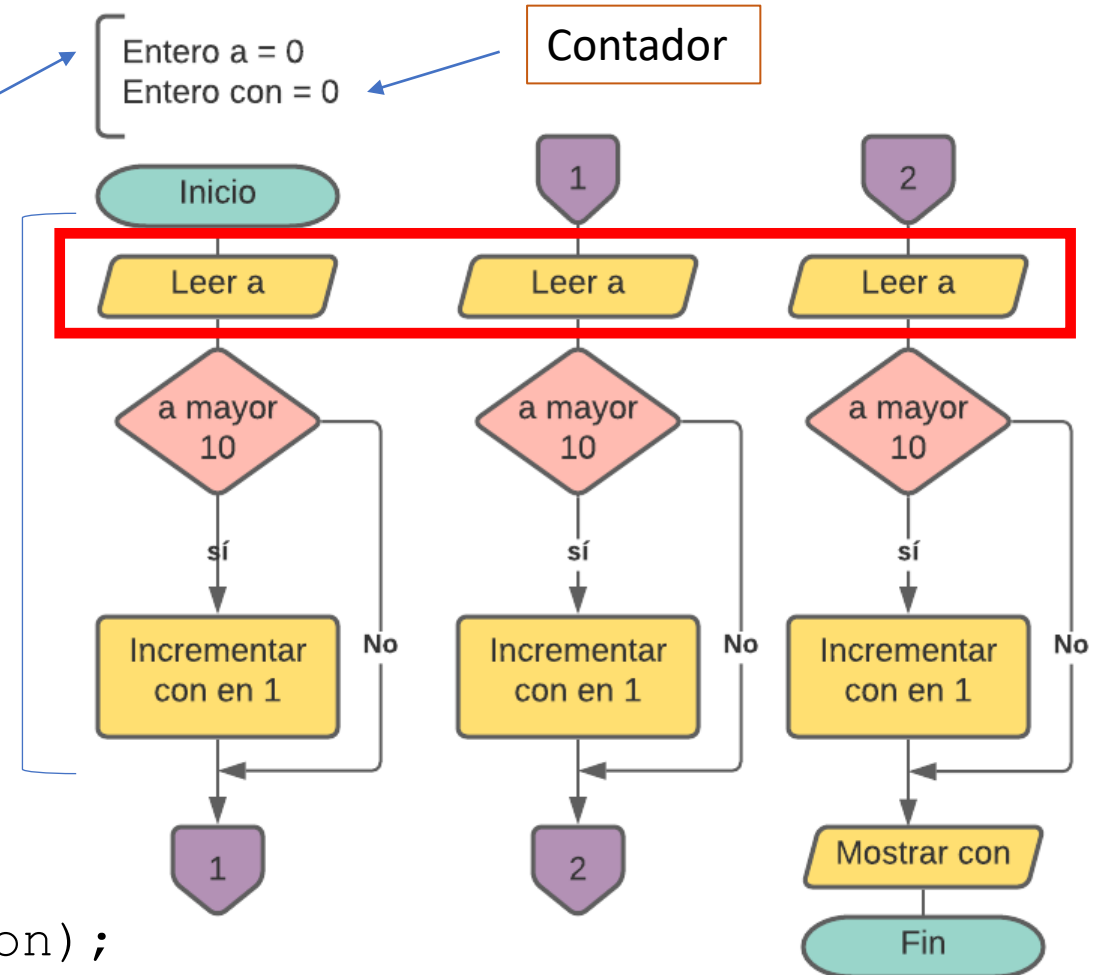
```
a = in.leerInt();
```

```
if(a>10) con++;
```

```
System.out.println("MAYORES QUE 10: "+con);
```

Utilizar una sola variable para leer cada dato y procesarlo

Proceso que se repite

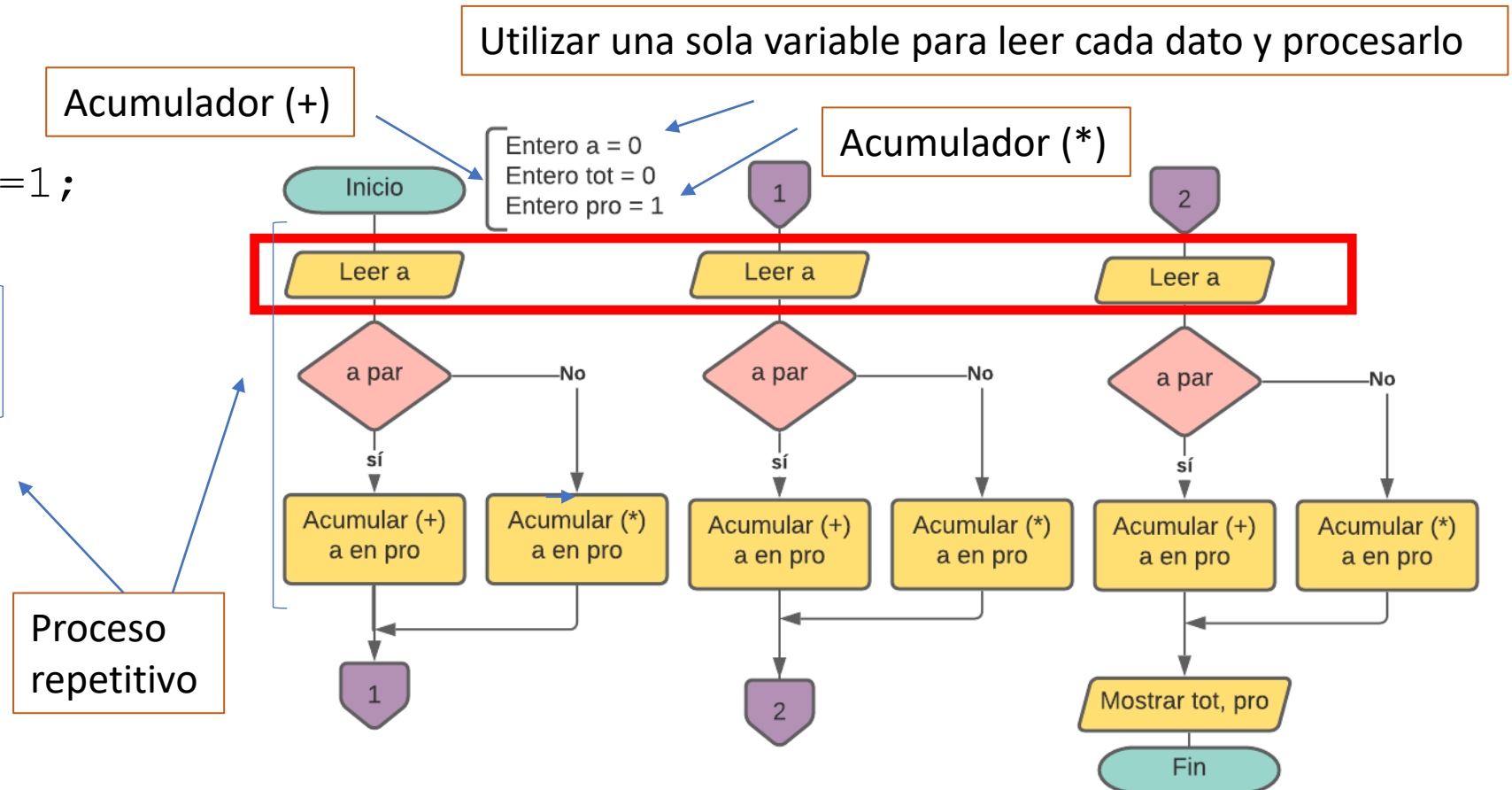


Acumuladores

Los acumuladores son totalizadores aditivos (+) y multiplicativos (*), inicializados a 0 y 1

Programa que lee tres enteros y obtiene la suma de los pares y el producto de los impares.

```
//Variables
int a=0, tot=0, pro=1;
//Programa
a = in.leerInt();
if(a%2==0) tot+=a;
else pro*=a;
a = in.leerInt();
if(a%2==0) tot+=a;
else pro*=a;
a = in.leerInt();
if(a%2==0) tot+=a;
else pro*=a;
System.out.println(tot+" "+pro);
```



Señales

Las señales son variables o condiciones V/F que informan si un proceso está completado o no

Programa que lee tres enteros y obtiene el primero de ellos que es positivo

//Variables

```
int a = 0, pp = -1;
```

//Programa

```
a = in.leerInt();
```

```
if(a>0 && pp<=0) pp=a;
```

```
a = in.leerInt();
```

```
if(a>0 && pp<=0) pp=a;
```

```
a = in.leerInt();
```

```
if(a>0 && pp<=0) pp=a;
```

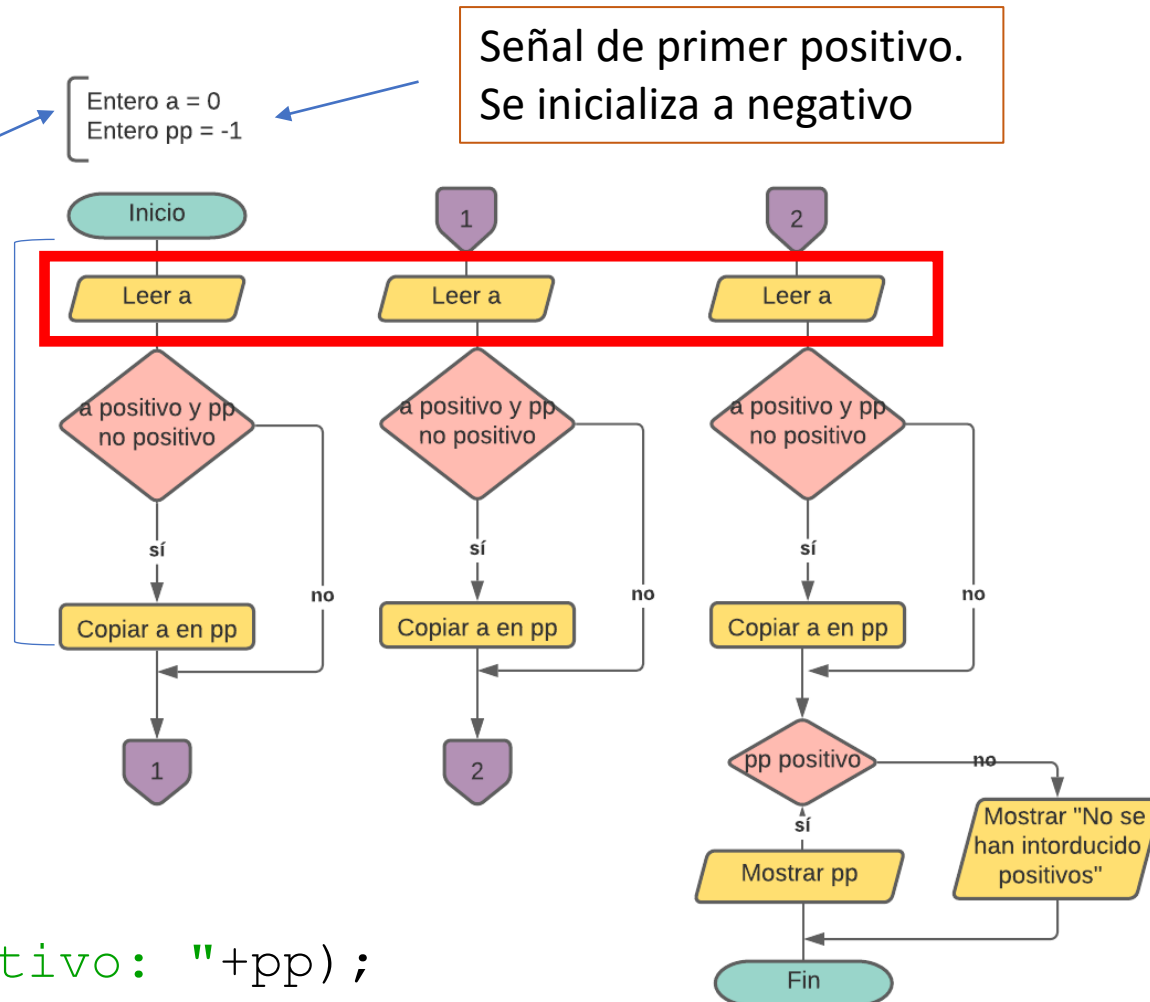
```
if(pp>0) System.out.println("Primer positivo: "+pp);
```

```
else System.out.println("No se han introducido positivos");
```

Utilizar una sola variable para leer cada dato y procesarlo

Proceso que se repite

Señal de primer positivo.
Se inicializa a negativo



Señales (II)

Uso de una variable booleana como señal

Programa que lee tres enteros y obtiene el primero de ellos que es positivo

//Variables

```
int a = 0, pp = -1;
```

//Programa

```
a = in.leerInt();
```

```
if(a>0 && !enc){pp=a;
```

```
enc = false;}
```

```
a = in.leerInt();
```

```
if(a>0 && !enc) ){pp=a;
```

```
enc = false;}
```

```
a = in.leerInt();
```

```
if(a>0 && !enc) ){pp=a;
```

```
enc = false;}
```

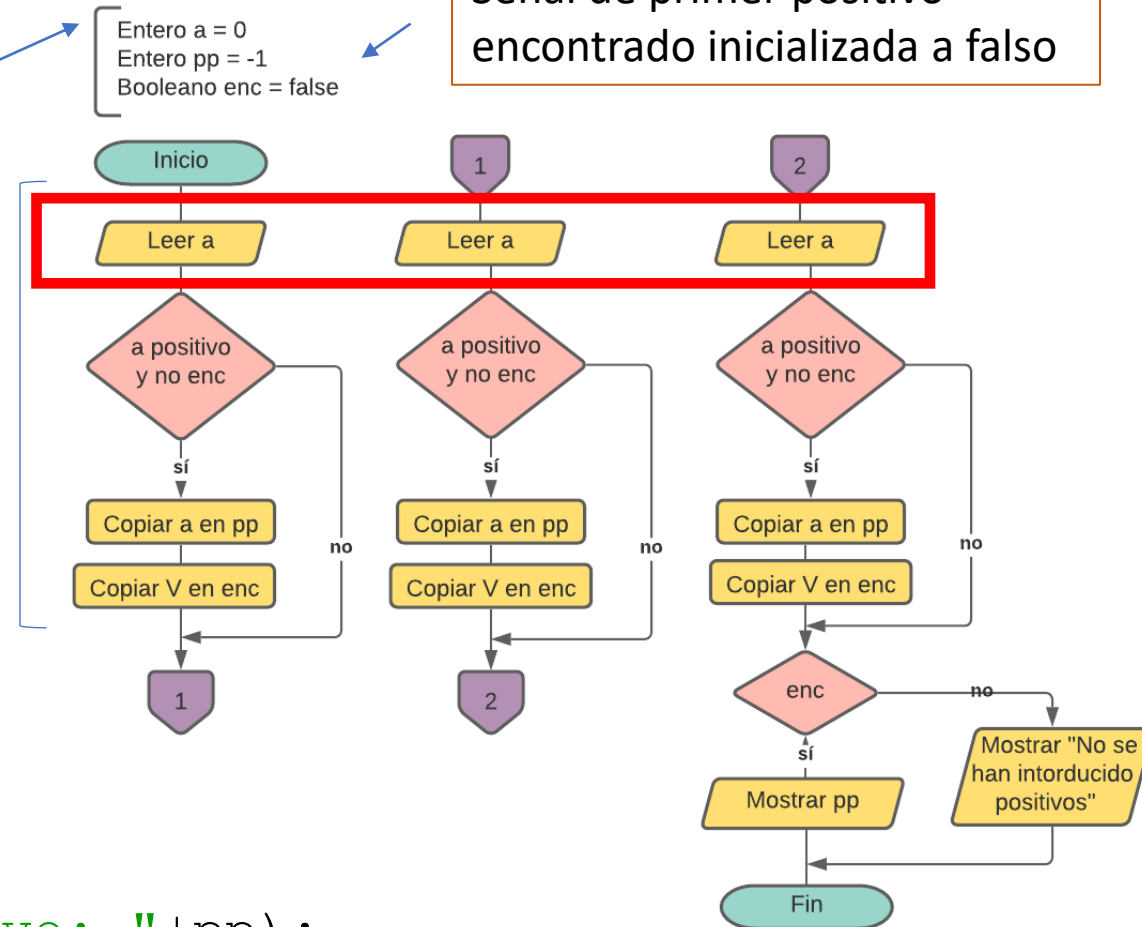
```
if(enc) System.out.println("Primer positivo: "+pp);
```

```
else System.out.println("No se han introducido positivos");
```

Utilizar una sola variable para leer cada dato y procesarlo

Proceso que se repite

Señal de primer positivo encontrado inicializada a falso



Ámbito de validez de una variable

//Variables

int a = 0;

//Programa

if (a%2==0)

{

int b = 1;

if (a<20)

{

int c = 3;

a += b + c;

}

if (a<10)

{

int c = 30;

a += b + c;

}

a++;

}

a++;

//Variables

int a = 0;

//Programa

if (a<20)

{

int b = 1;

if (a<10)

{

int c = 3;

a += b + c;

}

a += b;

}

a++;

- En un bloque de código se pueden declarar e inicializar variables (pueden haber variables con el mismo nombre en bloques distintos)
- Una variable es accesible sólo dentro de su bloque de código a partir de donde se declara (**ámbito de validez**)

En cada programa los bloques de código están indicados con los colores rojo, verde y azul, al igual que el ámbito de validez de cada variable

Tabla de ejecución ámbito de validez

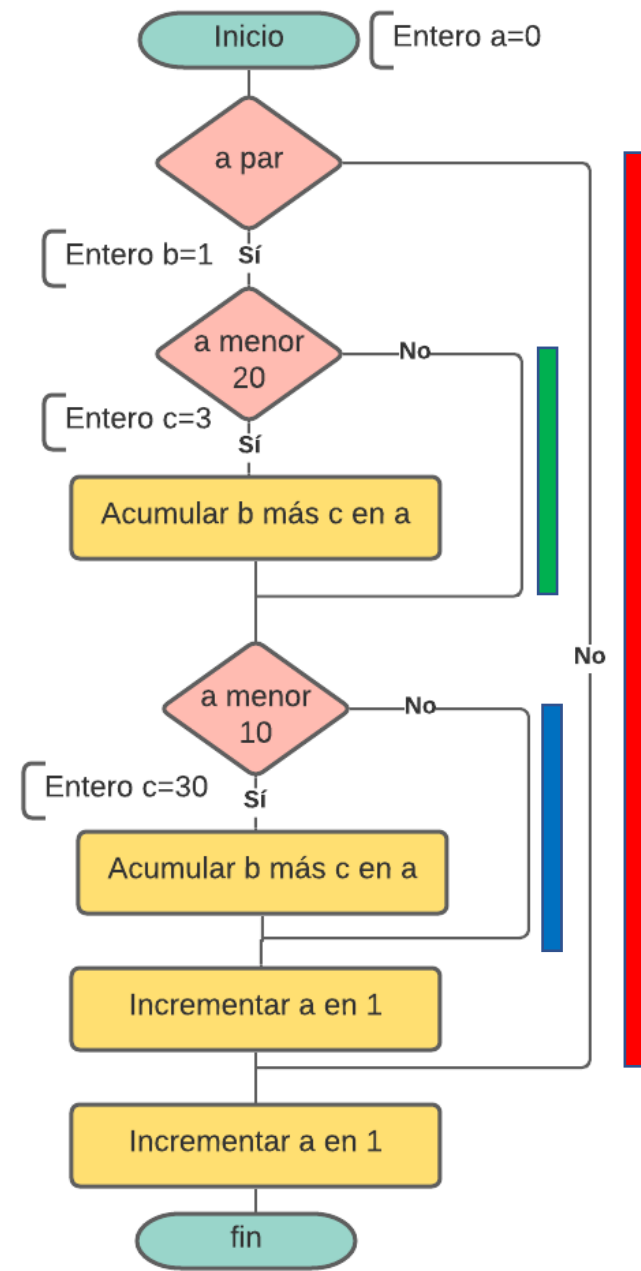
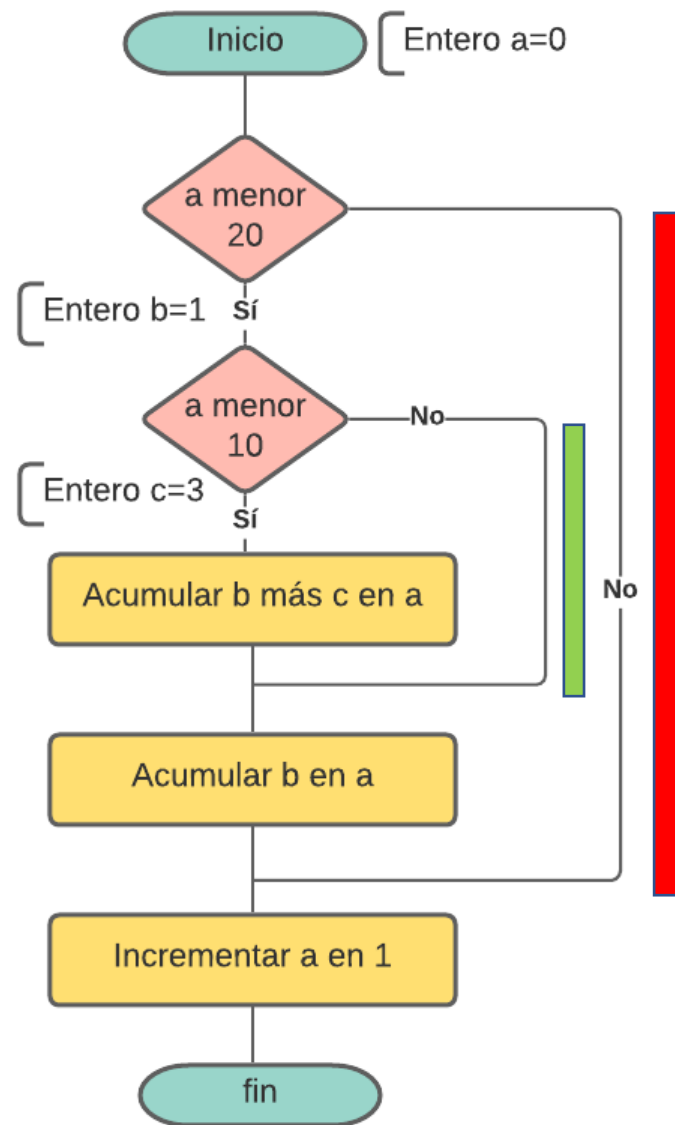
Hágase una tabla de ejecución de los programas anteriores

Ins/Con	Con	a	b	c
Inicializacion		0		
a<20	true	0		
Inicializacion		0	1	
a<10	true	0	1	
Inicializacion		0	1	3
a+=b+c		4	1	3
a+=b		5	1	
a++		6		

Ins/Con	Con	a	b	c
Inicializacion		0		
a%2==0	true	0		
Inicializacion		0	1	
a<20	true	0	1	
Inicialización		0	1	3
a+=b+c		4	1	3
a<10	true	4	1	
Inicialización		4	1	30
a+=b+c		35	1	3
a++		36	1	
a++		37		

Diagrama de flujo ámbito de validez

Hágase el diagrama de flujo para cada uno de los programas definidos en “Ámbito de validez de una variable”



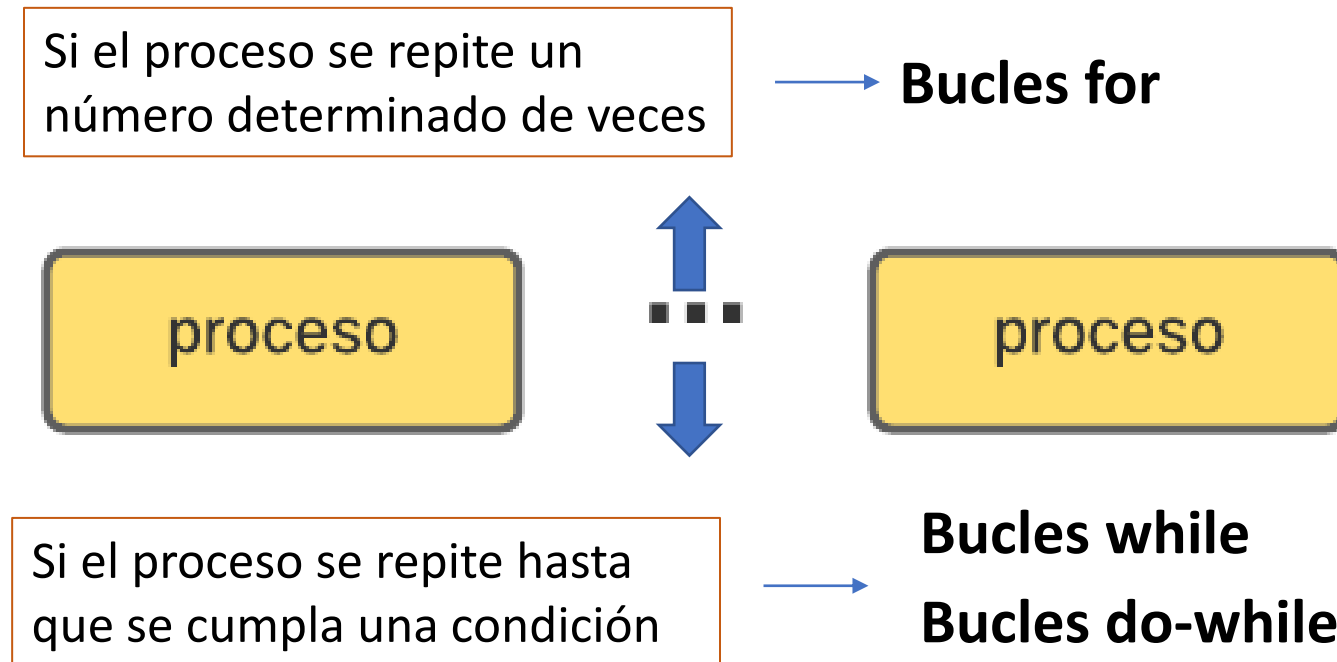
Instrucciones de bucle

Objetivos

- 1) Bucle while
- 2) Bucle do-while
- 3) Técnica para obtener la salida como una cadena
- 4) Bucle for
- 5) Saltos
- 6) Ciclos
- 7) Anidamiento de bucles
- 8) Etiquetas y salto

Instrucciones de bucles

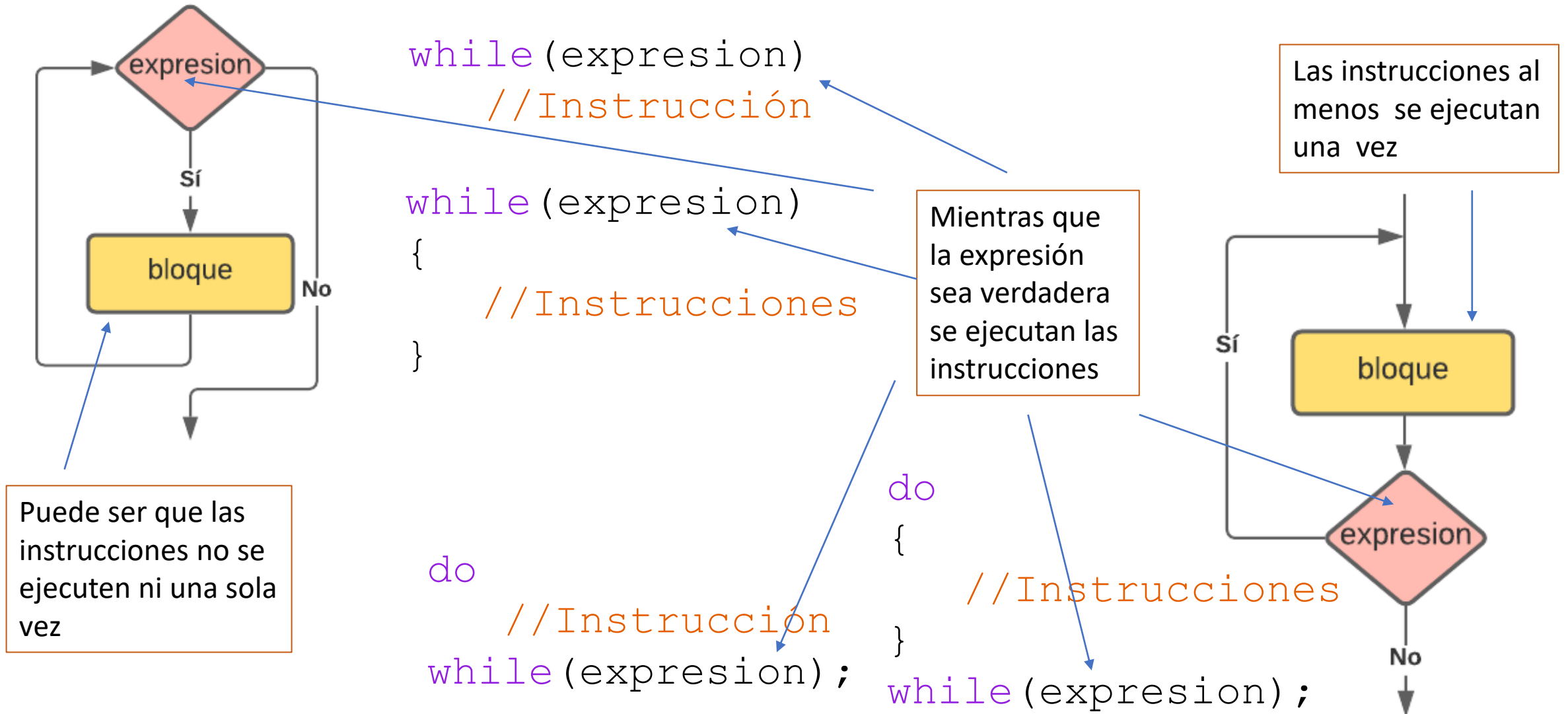
Las instrucciones de bucles son instrucciones que permiten que un proceso se repita un número determinado de veces o hasta que se cumpla una condición



Estrategias para conseguir procesos repetitivos

- Utilizar el menor número de variables de lectura
- Pensar en algoritmos que hacen el proceso “poco” a “poco” descomponiendo el problema en otro similar pero con datos más simples

Bucles while y do-while



Bucle while

Hágase un programa que lea notas entre 0 y 10 hasta introducir el 0. Se mostrará la media de las notas positivas redondeadas a un decimal

Variable para leer todas las notas

Datos

Entero a = 0
Entero total = 0
Entero con = 0
Real media = 0

Almacena el
número de
notas leídas

Almacena la suma
de todas las notas

Almacena la media
de las notas

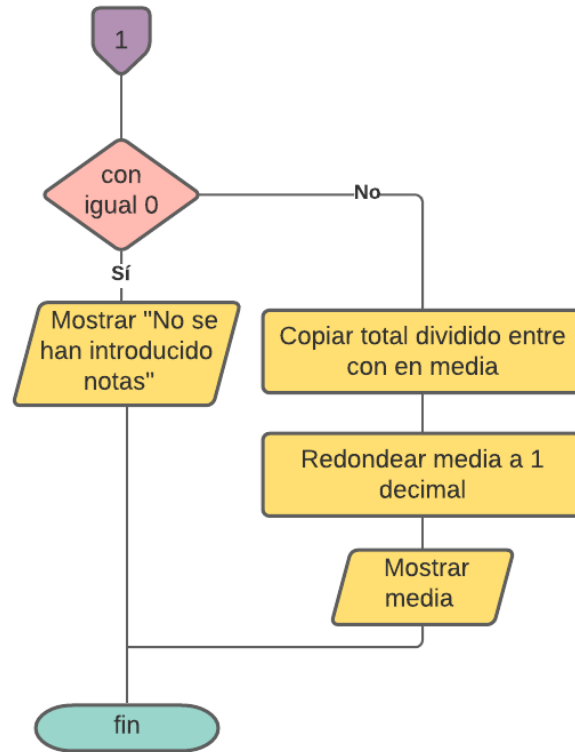
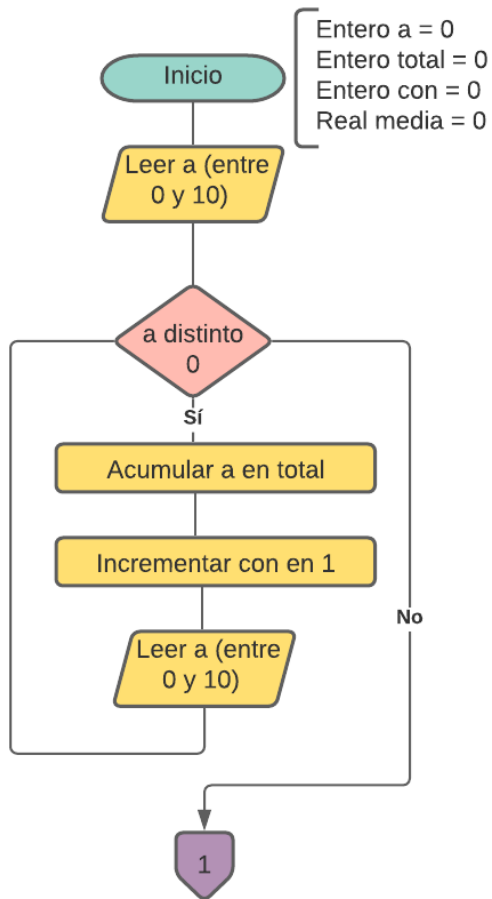
Condición mientras

Si "a" es distinto de 0

Algoritmo

1. Leer "a" (entre 0 y 10)
2. Si "a" es distinto de 0
 1. Acumular "a" en "total"
 2. Incrementar "con" en 1
 3. Leer "a" (entre 0 y 10)
3. Si "con" es 0 (no se han introducido datos)
 1. Mostrar que no se han introducido notas
4. Si no (al menos hay una nota)
 1. Copiar "total" dividido entre "con" en "media"
 - 2 Redondear "media" a 1 decimal
 - 2 Mostrar "media"

Bucle while (a)



```
//Variables
int a = 0;
int tot = 0;
int con = 0;
double media = 0;
```

//Programa

```
a = in.leerInt("Nota: ", v->v>=0 && v<=10);
while(a!=0)
{
    tot+=a;
    con++;
    a = in.leerInt("Nota: ", v->v>=0 && v<=10);
}
if(con==0)
    System.out.println("No se han introducido notas");
else
{
    media = tot/(double)con;
    media = (int)(media*10)/10.0;
    System.out.println(media);
}
```

Salida
consola

Nota: 8
Nota: 7
Nota: 4
Nota: 0
6.3

Tabla de ejecución bucle while

Hágase una tabla de ejecución del programa anterior introduciendo como notas 8, 7 y 4

Instrucción/Condición	Consola/Condición	a	tot	con	media
Inicializacion		0	0	0	0
a = in.leerInt("Nota: ")	Nota: 8	8	0	0	0
a!=0	true	8	0	0	0
tot+=a		8	8	0	0
con++		8	8	1	0
a = in.leerInt("Nota: ")	Nota: 7	7	8	1	0
a!=0	true	7	8	1	0
tot+=a		7	15	1	0
con++		7	15	2	0
a = in.leerInt("Nota: ")	Nota: 4	4	15	2	0



Tabla de ejecución bucle while

Instrucción/Condición	Consola/Condición	a	tot	con	media
		4	15	2	0
a!=0	true	4	15	2	0
tot+=a		4	19	2	0
con++		4	19	3	0
a = in.leerInt("Nota: ")	Nota: 0	0	19	3	0
a!=0	false	0	19	3	0
con==0	false	0	19	3	0
media = tot/(double)con	false	0	19	3	6,333333
media = (int)(media*10)/10.0		0	19	3	6,3
System.out.println(media)	6.3	0	19	3	6,3



Bucle do-while

Hágase un programa que genere números aleatorios entre -10 y 10 hasta generar el 0. El programa mostrará cada dato generado y los sumará

Datos

Entero a = 0
Entero total = 0

Variable donde se generan cada número aleatorio

Condición mientras

Si "a" distinto de 0

Almacena la suma de los números aleatorios generados

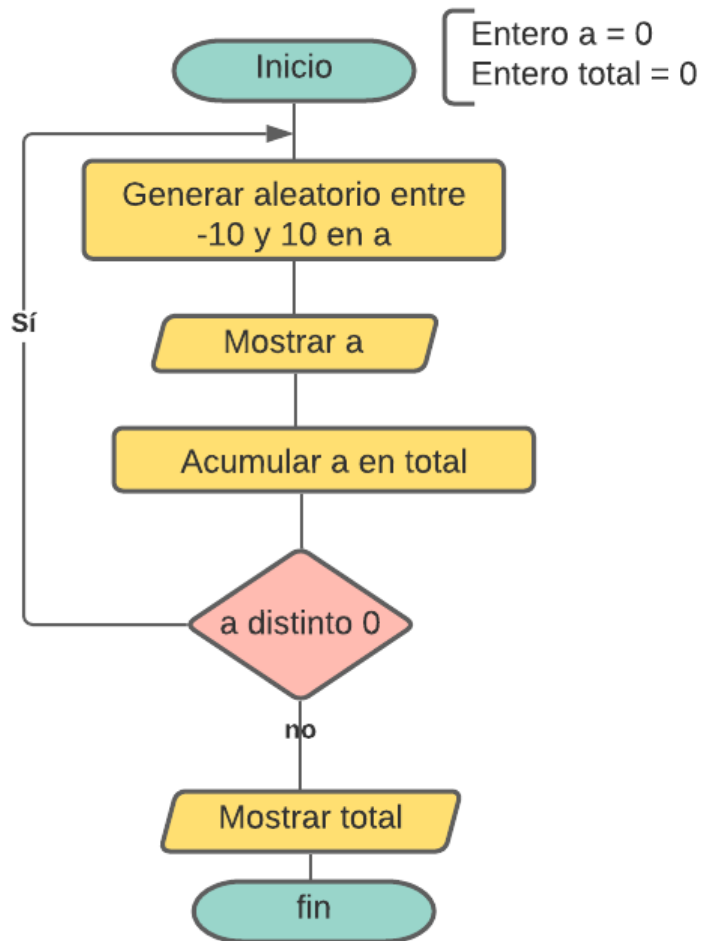
El proceso se repite al menos una vez

Algoritmo

1. Generar aleatorio entre -10 y 10 en "a"
2. Mostrar "a"
3. Acumular "a" en "total"
4. Si "a" distinto de 0
5. Mostrar "total"



Bucle do-while (II)



//Variables

```
int a = 0;  
int total = 0;
```

//Programa

```
do  
{  
    a = (int) (Math.random()*21) - 10;  
    System.out.println(a);  
    total+=a;  
} while (a!=0);
```

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

Salida
consola

6
5
5
-1
-2
-2
4
0
15

Tabla de ejecución bucle do-while

Hágase una tabla de ejecución del programa anterior

Instrucción/Condición	Consola/Condición	a	total
Inicializacion		0	0
a = (int) (Math.random()*21) - 10		8	0
System.out.println(a)	8	8	0
total+=a		8	8
a!=0	true	8	8
a = (int) (Math.random()*21) - 10		-6	8
System.out.println(a)	-6	-6	8
total+=a		-6	2
a!=0	true	-6	2



Tabla de ejecución bucle do-while

Instrucción/Condición	Consola/Condición	a	total
		-6	2
a = (int) (Math.random()*21) - 10		0	2
System.out.println(a)	0	0	2
total+=a		0	2
a!=0	false	0	2
System.out.println(total)	2		



Técnica obtener la salida en una cadena

Una técnica para no tener que mezclar los datos de lectura con los datos de salida consiste en acumular en una variable de cadena la salida con los saltos de línea ('\n') y formato (uso del método "format")

Hágase un programa que lea enteros no negativos entre 1 y 999, ambos inclusive, hasta introducir el 0 o un máximo de 20 datos. El programa mostrará un informe de las siguientes características:

NUMERO	SUBTOTAL
007	7
108	117
067	182

Como entrada se	Numero: 7
introdujeron los	Numero: 108
siguientes valores:	Numero: 67
	Numero: 0

Técnica obtener la salida en una cadena (a)

Variable para leer cada dato

Datos

int n = 0

int tot = 0

Int con = 0

String s = ""

Cálculo de
subtotales

Variable de
salida

Contar número
de datos

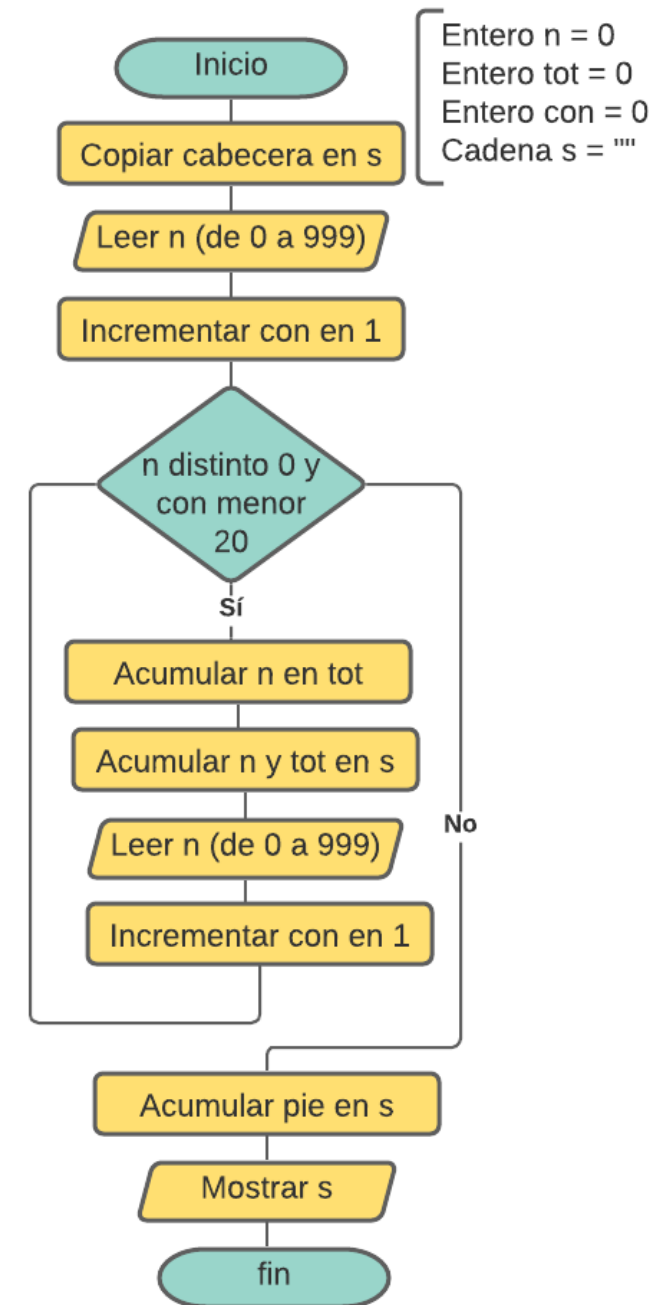
Condición mientras

Si n distinto de 0 y con
menor que 20

Algoritmo

1. Copiar la cabecera en "s"
2. Leer "n" (de 0 a 999)
3. Incrementar "con" en 1
4. Si "n" distinto de 0 y "con" menor que 20

1. Acumular "n" en "tot"
2. Acumular "n" y "tot" en "s" con formato
3. Leer "n" (0 a 999)
4. Incrementar con en 1
5. Acumular pie en "s"
6. Mostrar "s"




```
//Variables
```

```
int n = 0, tot = 0, con = 0;  
String s = "";
```

```
//Programa
```

```
s = "NUMERO SUBTOTALES\n"+  
    "===== \n";
```

```
n = in.leerInt("Numero: ", v->v>=0 && v<1000);
```

```
con++;
```

```
while (n!=0 && con<20)
```

```
{
```

```
    tot+=n;
```

```
    s+=String.format("    %03d %20d\n", n, tot);
```

```
    n = in.leerInt("Numero: ", v->v>=0 && v<1000);
```

```
    con++;
```

```
}
```

```
s+="===== \n";
```

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



Salida
consola

Numero: 7

Numero: 108

Numero: 67

Numero: 0

NUMERO SUBTOTALES

```
=====
007                                7
108                               115
067                               182
=====
```

Técnica obtener
la salida en una
cadena (b)

Bucles while y do-while

Hágase un programa que lea un entero positivo y sume todos sus dígitos

Almacena el entero a leer, cuyos dígitos se quiere sumar

Datos

Entero n = 0

Entero total = 0

Entero copia = 0;

Entero u = 0

Almacena la suma de todos los dígitos

Almacena una copia de "n"

Almacena las unidades del valor actual de copia

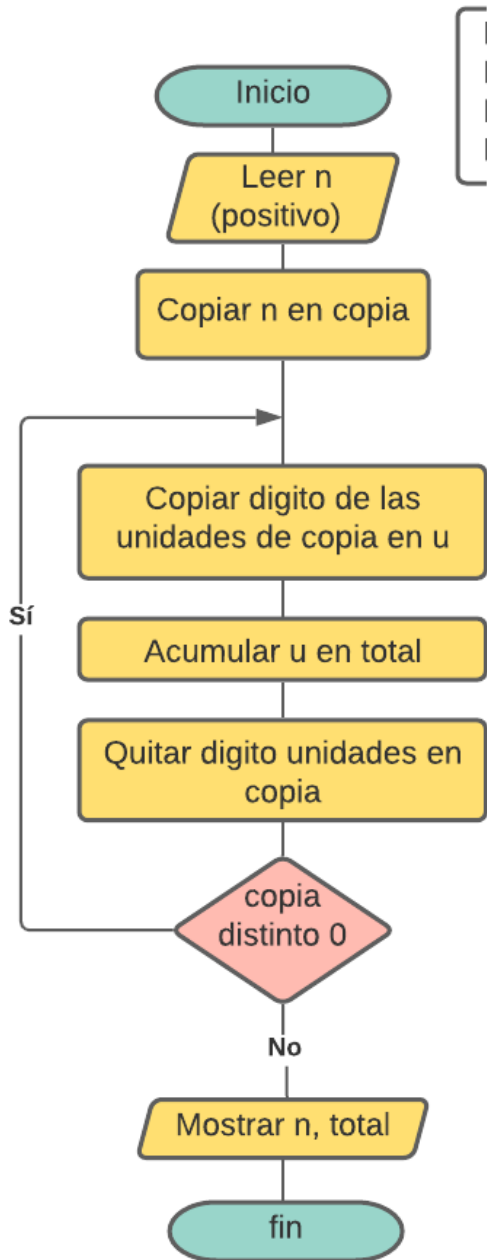
Estrategia Obtener y quitar el dígito de las unidades del número leído hasta que sea 0. Como al final se tiene que mostrar el número leído, se hará una **copia** y se aplica la estrategia sobre la copia

Condición mientras
Si "copia" es distinto de 0

Algoritmo

1. Leer "n" (positivo)
2. Copiar "n" en "copia"
3. Copiar el dígito de las unidades de "copia" en "u"
4. Acumular "u" en "total"
5. Quitar el dígito de las unidades en "copia"
6. Si "copia" distinto de 0
7. Mostrar "n" y "total"

Bucles while y do-while



Entero n = 0
Entero total = 0
Entero u = 0
Entero copia = 0

//Variables

```
int n = 0;
int total = 0;
int u = 0;
int copia = 0;
```

//Programa

```
n = in.leerInt("Entero: ", v->v>0);
copia = n;
```

do

```
{
    u = copia % 10;
    total+=u;
    copia = copia / 10;
} while(copia!=0);
```

```
System.out.println("La suma de los dígitos de "+n+" es " + total);
```

Entero: 1085

La suma de los dígitos de 1085 es 14

Salida
consola

Tabla de ejecución bucle while y do-while

Hágase una tabla de ejecución del programa anterior introduciendo como número el 1085

Instrucción/Condición	Consola/Condición	n	total	copia	u
Inicializacion		0	0	0	0
n= in.leerInt("Entero: ")	Entero: 1085	1085	0	0	0
copia=n		1085	0	1085	0
u=copia%10		1085	0	1085	5
total+=u		1085	5	1085	5
copia=copia/10		1085	5	108	5
copia!=0	true	1085	5	108	5
u=copia%10		1085	5	108	8
total+=u		1085	13	108	8
copia=copia/10		1085	13	10	8
copia!=0	true	1085	13	10	8



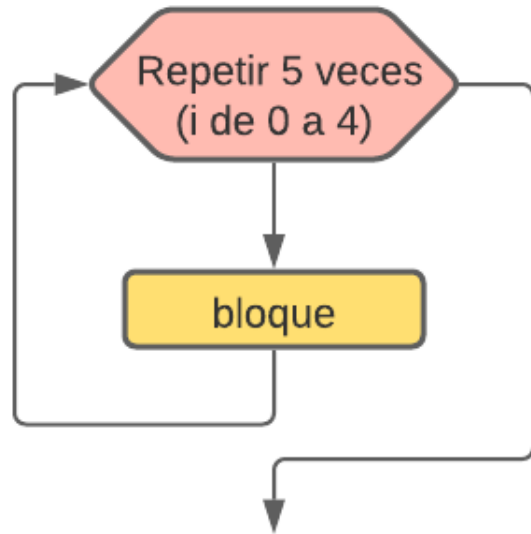


Tabla de ejecución bucle while y do-while

Instrucción/Condición	Consola/Condición	n	total	copia	u
		1085	13	10	8
u=copia%10		1085	13	10	0
total+=u		1085	13	10	8
copia=copia/10		1085	13	1	8
copia!=0	true	1085	13	1	8
u=copia%10		1085	13	1	1
total+=u		1085	14	1	1
copia=copia/10		1085	14	0	1
copia!=0	false	1085	14	0	1
System.out.println("La suma de los dígitos de " + n + " es "+total);	La suma de los dígitos de 1085 es 14				

Bucle for



//Ascendente

```
for ( int i=0 ; i<5 ; i++ )
```

Declaración e inicialización de la variable de recorrido. Sólo es accesible en el bloque

Se actualiza la variable de recorrido. Puede modificarse con otras expresiones como $i+=2$, etc

Si la expresión es verdadera se ejecutan las instrucciones

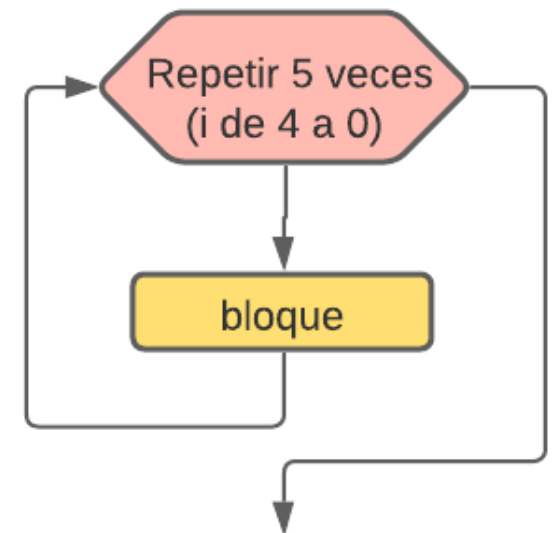
```
//Instrucciones
```

```
}  
  
for (int i=0; i<5; i++)  
    //Instrucción
```

//Descendente

```
for (int i=4; i>=0; i--)  
{  
    //Instrucciones  
}
```

```
for (int i=4; i>=0; i--)  
    //Instrucción
```



Bucle for (II)

Hágase un programa que genere 5 números aleatorios entre 0 y 9, ambos inclusive. Se mostrarán los datos en una línea separados por comas y se obtendrá el menor de ellos

Variable donde se generan cada número aleatorio

Datos

Entero a = 0

Entero menor = 9

Almacena el menor de los datos generados. Se inicializa al mayor valor posible (9)

Condición for

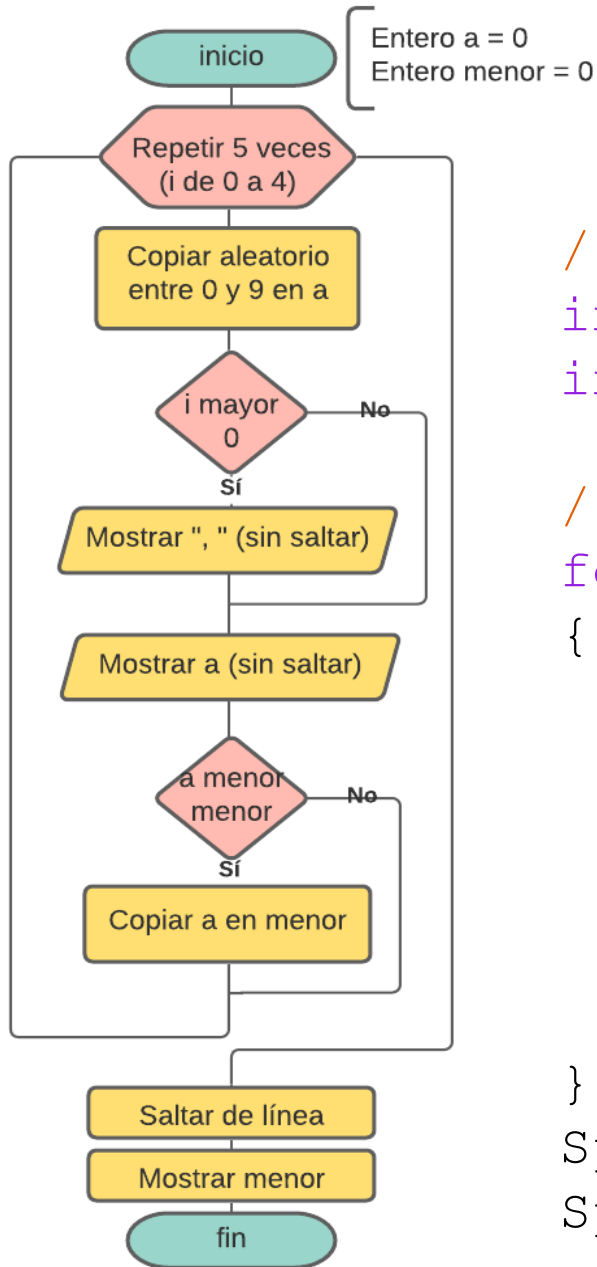
Para "i" de 0 a 4

Acceso a la variable de recorrido dentro del bloque

Algoritmo

1. Para "i" de 0 a 4
 1. Generar aleatorio entre 0 y 9 en "a"
 2. Si "i" mayor que 0
 1. Mostrar "," (sin saltar)
 3. Mostrar "a" (sin saltar)
 4. Si "a" menor que "menor"
 1. Copiar "a" en "menor"
 2. Saltar de línea
 3. Mostrar "menor"

Bucle for (Ila)



//Variables

```
int a = 0;
```

```
int menor = 9;
```

//Programa

```
for(int i=0; i<5; i++)
```

```
{
```

```
    a = (int) (Math.random()*10);
```

```
    if(i>0) System.out.print(", ");
```

```
    System.out.print(a);
```

```
    if(a<menor) menor = a;
```

```
}
```

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

```
System.out.println("El menor es "+menor);
```

Salida
consola



4, 5, 8, 7, 8
El menor es 4

Tabla de ejecución bucle for

Hágase una tabla de ejecución del programa anterior

Instrucción/Condición	Consola/Condición	a	menor	i
Inicialización		0	9	
int i = 0		0	9	0
i<5	true	0	9	0
a = (int) (Math.random()*10);		4	9	0
i>0	false	4	9	0
System.out.print(a);	4	4	9	0
a<menor	true	4	9	0
menor = a		4	4	0
i++		4	4	1
i<5	true	4	4	1
a = (int) (Math.random()*10);		5	4	1




Tabla de ejecución bucle for

Instrucción/Condición	Consola/Condición	a	menor	i
		5	4	1
i>0	true	5	4	1
System.out.print(", ");	4,	5	4	1
System.out.print(a);	4, 5	5	4	1
a<menor	false	5	4	1
i++		5	4	2
i<5	true	5	4	2
a = (int) (Math.random()*10);		8	4	2
i>0	true	8	4	2
System.out.print(", ");	4, 5,	8	4	2
System.out.print(a);	4, 5, 8	8	4	2
a<menor	false	8	4	2

Tabla de ejecución bucle for

Instrucción/Condición	Consola/Condición	a	menor	i
		8	4	2
i++		5	4	3
i<5	true	5	4	3
a = (int) (Math.random()*10);		7	4	3
i>0	true	7	4	3
System.out.print(", ");	4, 5, 8,	7	4	3
System.out.print(a);	4, 5, 8, 7	7	4	3
a<menor	false	7	4	3
i++		7	4	4
i<5	true	7	4	4
a = (int) (Math.random()*10);		8	4	4
i>0	true	8	4	4

Tabla de ejecución bucle for

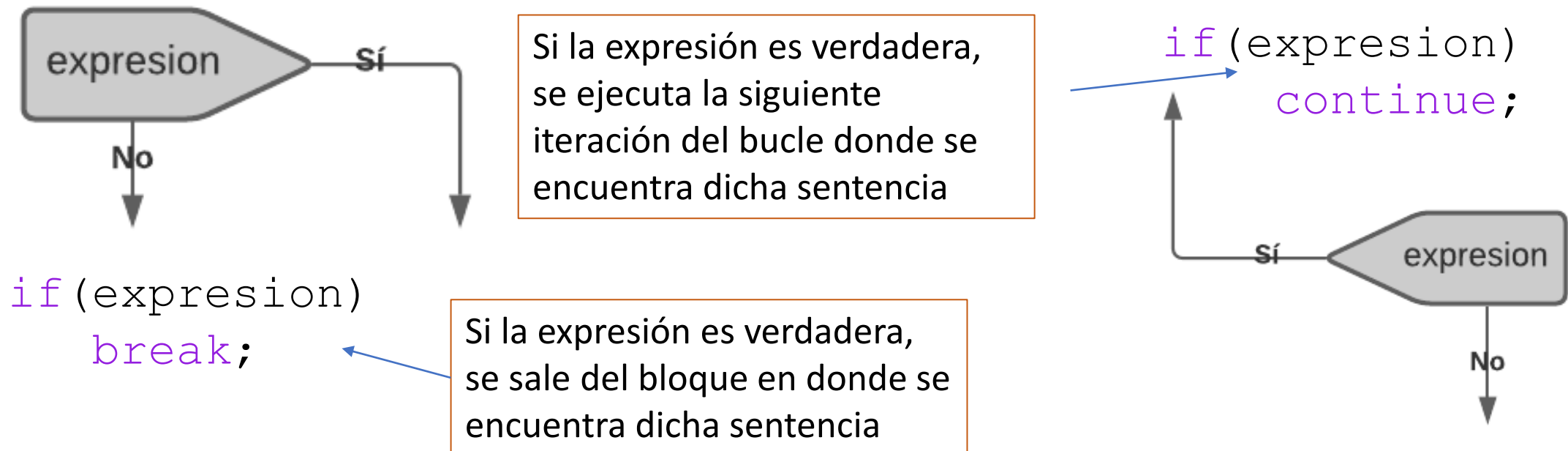


Instrucción/Condición	Consola/Condición	a	menor	i
		8	4	4
System.out.print(", ");	4, 5, 8, 7,	8	4	4
System.out.print(a);	4, 5, 8, 7, 8	8	4	4
a<menor	false	8	4	4
i++		8	5	5
i<5	false	8	5	5
System.out.println();		8	5	5
System.out.println("El menor es "+menor);	El menor es 4	8	5	5

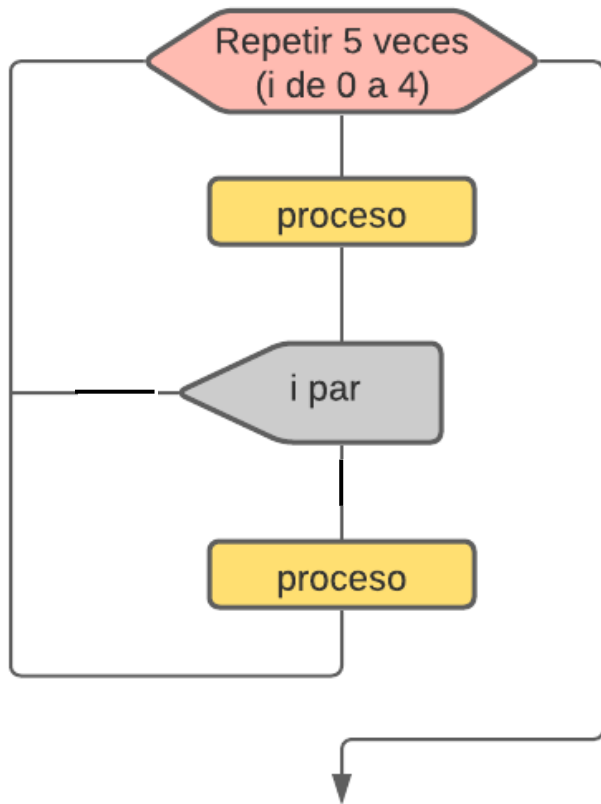
Salto

Las sentencias “break” y “continue” permiten realizar saltos en el código rompiendo la estructura de nuestro programa.

Suelen utilizarse con la instrucción “if”

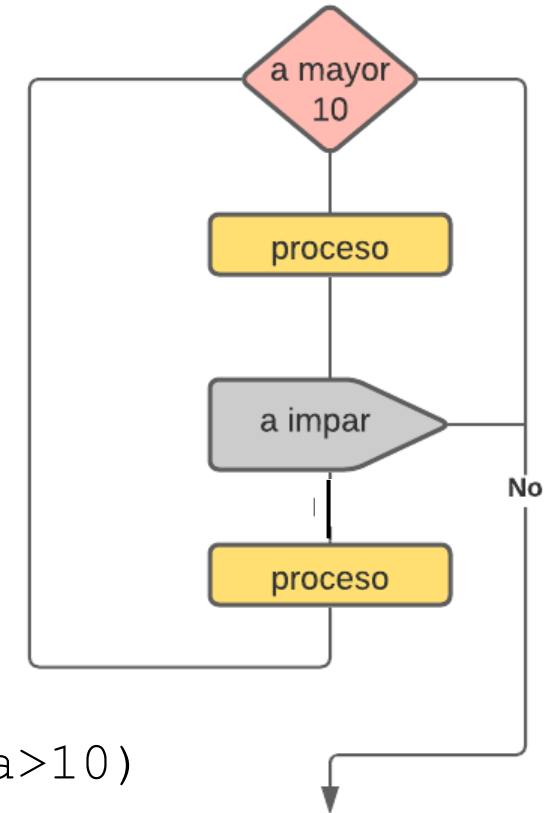


Salto (II)



```
for (int i=0; i<5; i++)  
{  
    //Instrucciones  
    if (i%2==0)  
        continue;  
    //Instrucciones  
}
```

A blue arrow points from the `continue;` statement in the code to the decision diamond in the flowchart.



```
while (a>10)  
{  
    //Instrucciones  
    if (a%2==1)  
        break;  
    //Instrucciones  
}
```

A blue arrow points from the `break;` statement in the code to the decision diamond in the flowchart.



Ciclos

```
while(true)
```

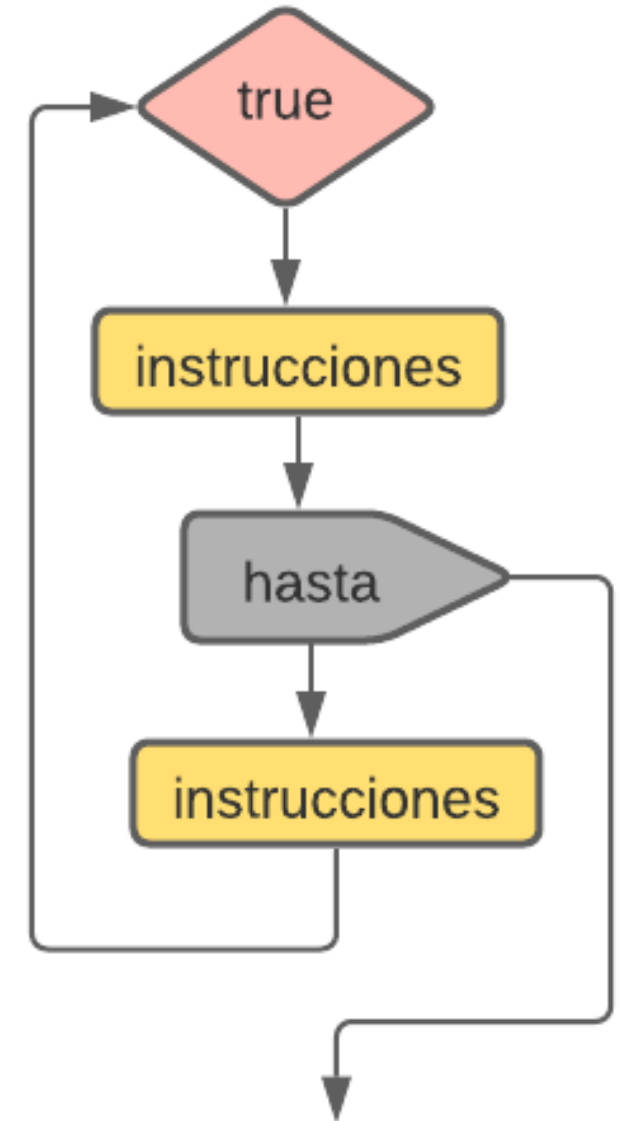
```
{
```

```
    //Instrucciones (al menos 1 vez) ↔
```

```
    if(hasta) break; ↔
```

```
    //Instrucciones (puede que no se  
    //ejecuten ninguna vez) ↔
```

```
}
```



Ciclos (II)

Hágase un programa que lea notas entre 0 y 10 hasta introducir el 0. Se mostrará la media de las notas positivas redondeadas a un decimal

Variable para leer todas las notas

La nota 0 no se tiene en cuenta para el cálculo de la media

Datos

Entero a = 0
Entero total = 0
Entero con = 0
Real media = 0

Condición hasta

Si "a" es igual a 0, salir

Algoritmo

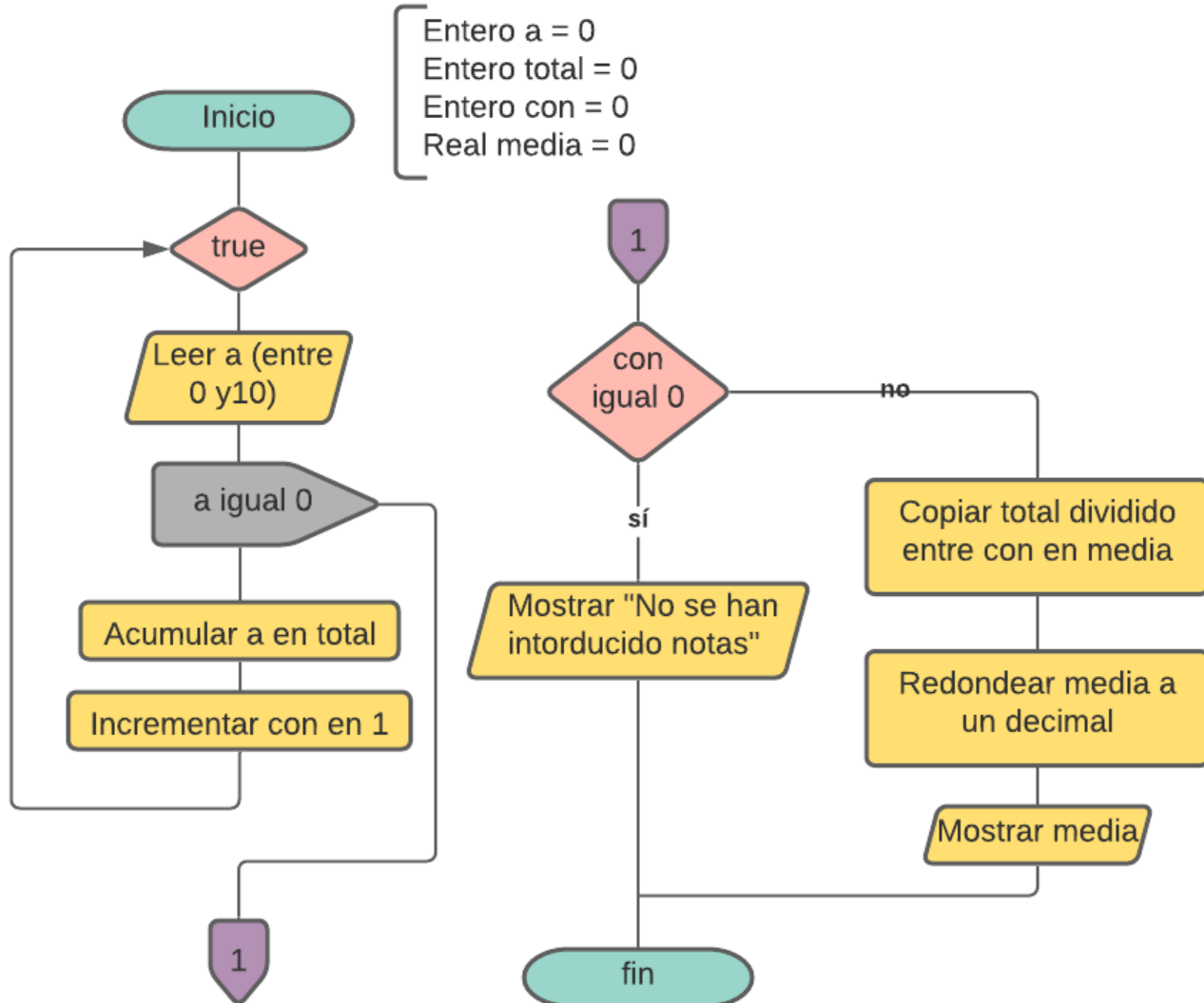
1. Leer "a" (entre 0 y 10)
2. Acumular "a" en "total"
3. Incrementar "con" en 1
4. Si "con" es 0 (no se han introducido datos)
 1. Mostrar que no se han introducido datos
5. Si no (al menos hay una nota)
 1. Copiar "total" dividido entre "con" en "media"
 - 2 Redondear "media" a 1 decimal
 - 2 Mostrar "media"

Almacena el
número de
notas leídas

Almacena la suma
de todas las notas

Almacena la media
de las notas

Ciclos (Ila)



//Variables

```
int a = 0;
```

```
int tot = 0;
```

```
int con = 0;
```

```
double media = 0;
```

//Programa

```
while(true)
```

```
{
```

```
    a = in.leerInt("Nota: ", v->v>=0 && v<=10);
```

```
    if(a==0) break;
```

```
    tot+=a;
```

```
    con++;
```

```
}
```

```
if(con==0)
```

```
    System.out.println("No se han introducido notas");
```

```
else
```

```
{
```

```
    media = tot/(double)con;
```

```
    media = (int)(media*10)/10.0;
```

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

```
}
```

Nota: 8

Nota: 7

Nota: 4

Nota: 0

Salida
consola

6.3

Tabla de ejecución ciclo

Hágase una tabla de ejecución del programa anterior introduciendo como notas 8, 7 y 4

Instrucción/Condición	Consola/Condición	a	tot	con	media
Inicializacion		0	0	0	0
a = in.leerInt("Nota: ")	Nota: 8	8	0	0	0
a==0	false	8	0	0	0
tot+=a		8	8	0	0
con++		8	8	1	0
a = in.leerInt("Nota: ")	Nota: 7	7	8	1	0
a==0	false	7	8	1	0
tot+=a		7	15	1	0
con++		7	15	2	0



Tabla de ejecución ciclo

Instrucción/Condición	Consola/Condición	a	tot	con	media
		7	15	2	0
a = in.leerInt("Nota: ")	Nota: 4	4	15	2	0
a==0	false	4	15	2	0
tot+=a		4	19	2	0
con++		4	19	3	0
a = in.leerInt("Nota: ")	Nota: 0	0	19	3	0
a==0	true	0	19	3	0
con==0	false	0	19	3	0
media = tot/(double)con	false	0	19	3	6,333333
media = (int)(media*10)/10.0		0	19	3	6,3
System.out.println(media)	6.3	0	19	3	6,3

Ciclos (III)

Hágase un programa que lea un entero positivo y sume todos sus dígitos

Almacena el entero a leer, cuyos dígitos se quiere sumar

Datos

Entero $n = 0$

Entero total = 0

Entero copia = 0;

Entero $u = 0$

Almacena una copia de "n"

Almacena la suma de todos los dígitos

Almacena las unidades del valor actual de copia

Condición hasta

Si "copia" es igual a 0, salir

Estrategia: En copia se ira quitando el dígito de las unidades hasta que sea 0

Algoritmo

1. Leer "n" (positivo)
2. Copiar "n" en "copia"
3. Copiar el dígito de las unidades de "copia" en "u"
4. Acumular "u" en "total"
5. Quitar el dígito de las unidades en "copia"
6. Mostrar "n" y "total"



Ciclos (IIa)

//Variables

```
int n = 0;  
int total = 0;  
int copia = 0;  
int u = 0;
```

//Programa

```
n = in.leerInt("Entero: ", v->v>0);  
copia = n;  
while(true)  
{  
    u = copia % 10;  
    total += u;  
    copia = copia /10;  
    if(copia==0) break;  
}
```

```
System.out.println("La suma de los dígitos de " + n + " es "+total);
```

Entero: 1085

La suma de los dígitos de 1085 es 14

Salida
consola

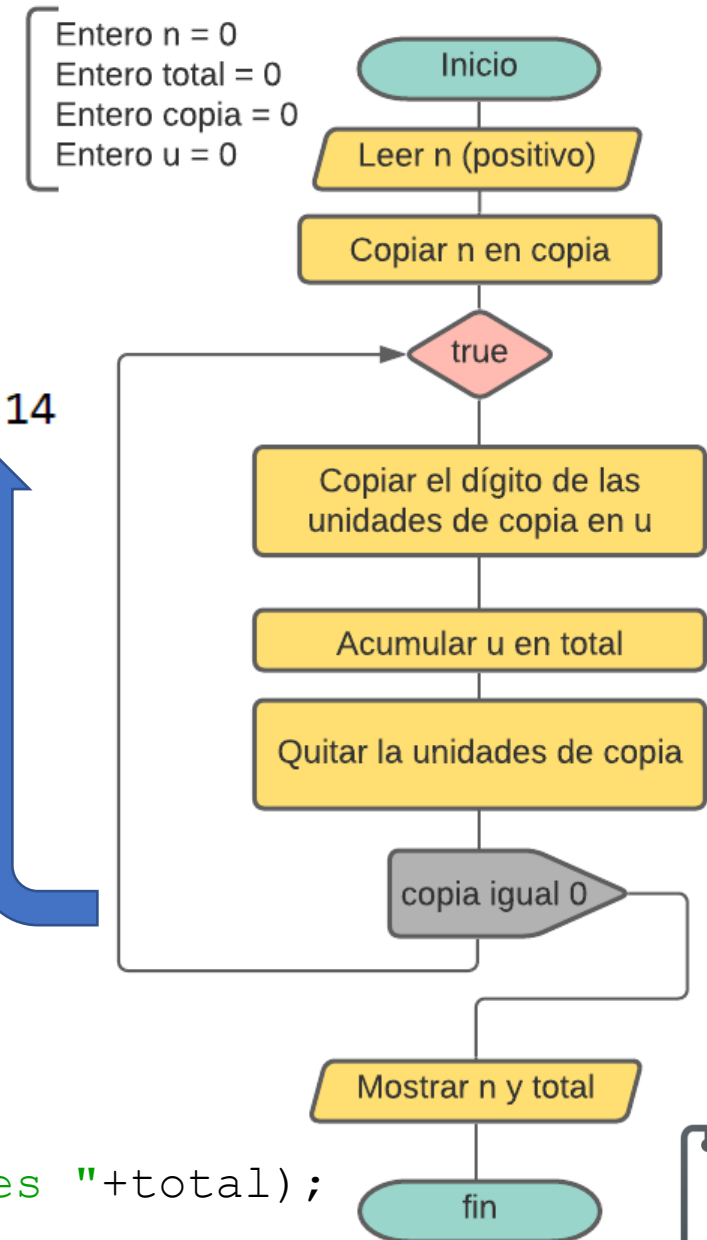



Tabla de ejecución ciclo

Hágase una tabla de ejecución del programa anterior introduciendo como número el 1085

Instrucción/Condición	Consola/Condición	n	total	copia	u
Inicializacion		0	0	0	0
n= in.leerInt("Entero: ")	Entero: 1085	1085	0	0	0
copia=n		1085	0	1085	0
u=copia%10		1085	0	1085	5
total+=u		1085	5	1085	5
copia=copia/10		1085	5	108	5
copia==0	false	1085	5	108	5
u=copia%10		1085	5	108	8
total+=u		1085	13	108	8
copia=copia/10		1085	13	10	8
copia==0	false	1085	13	10	8



Tabla de ejecución ciclo

Instrucción/Condición	Consola/Condición	n	total	copia	u
		1085	13	10	8
u=copia%10		1085	13	10	0
total+=u		1085	13	10	8
copia=copia/10		1085	13	1	8
copia==0	false	1085	13	1	8
u=copia%10		1085	13	1	1
total+=u		1085	14	1	1
copia=copia/10		1085	14	0	1
copia==0	true	1085	14	0	1
System.out.println("La suma de los dígitos de " + n + " es "+total);	La suma de los dígitos de 1085 es 14				

Anidamiento de bucles

Dentro de un bucle puede haber otro bucle

Bucle for

Bucle while

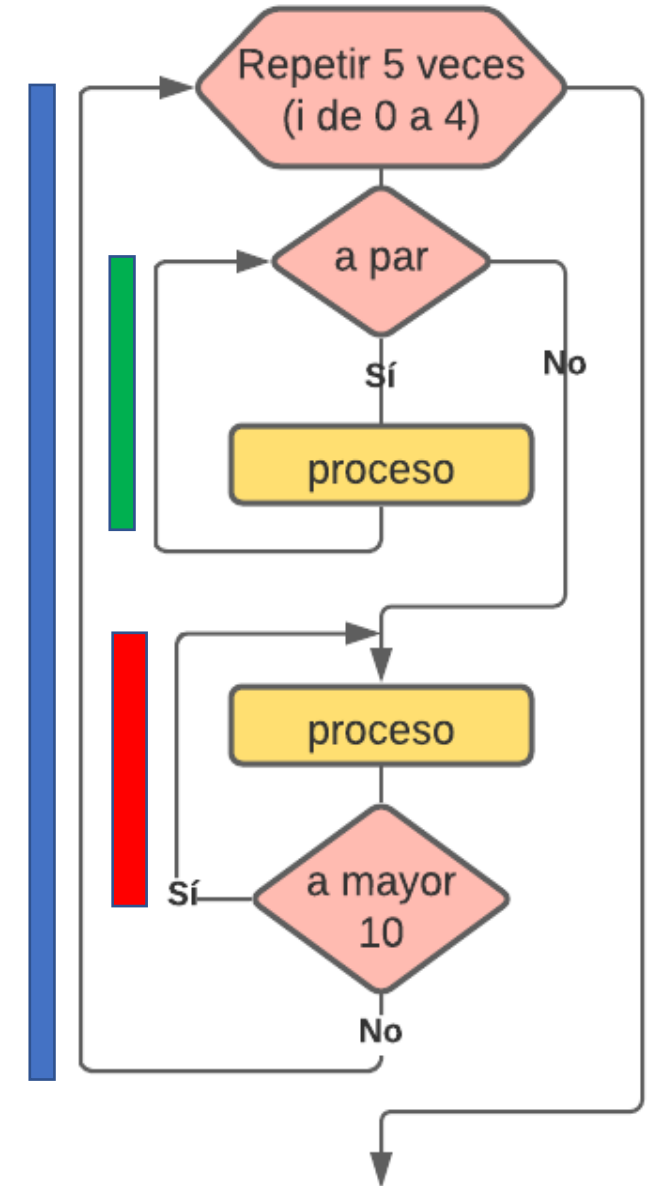
Bucle do-while

Bucle for

Bucle while

Bucle do-while

```
for (int i=0; i<5; i++)  
{  
    while (a%2==0)  
    {  
          
    }  
    do  
    {  
          
    } while (a>10);  
}
```



Anidamiento de bucles

Hágase un programa que lea enteros positivos hasta introducir el cero y muestre los divisores de cada uno de los números introducidos.

Variable que almacena cada numero leído

Datos

Entero $a = 0$

Salida $s = ""$

Variable que almacena la salida

Condición mientras

Si "a" distinto de 0

Algoritmo

1. Leer "a" (positivo o cero)
2. Si "a" distinto de 0
 1. DIVISORES DE "a"
 2. Leer "a" (positivo o cero)
3. Mostrar "s"

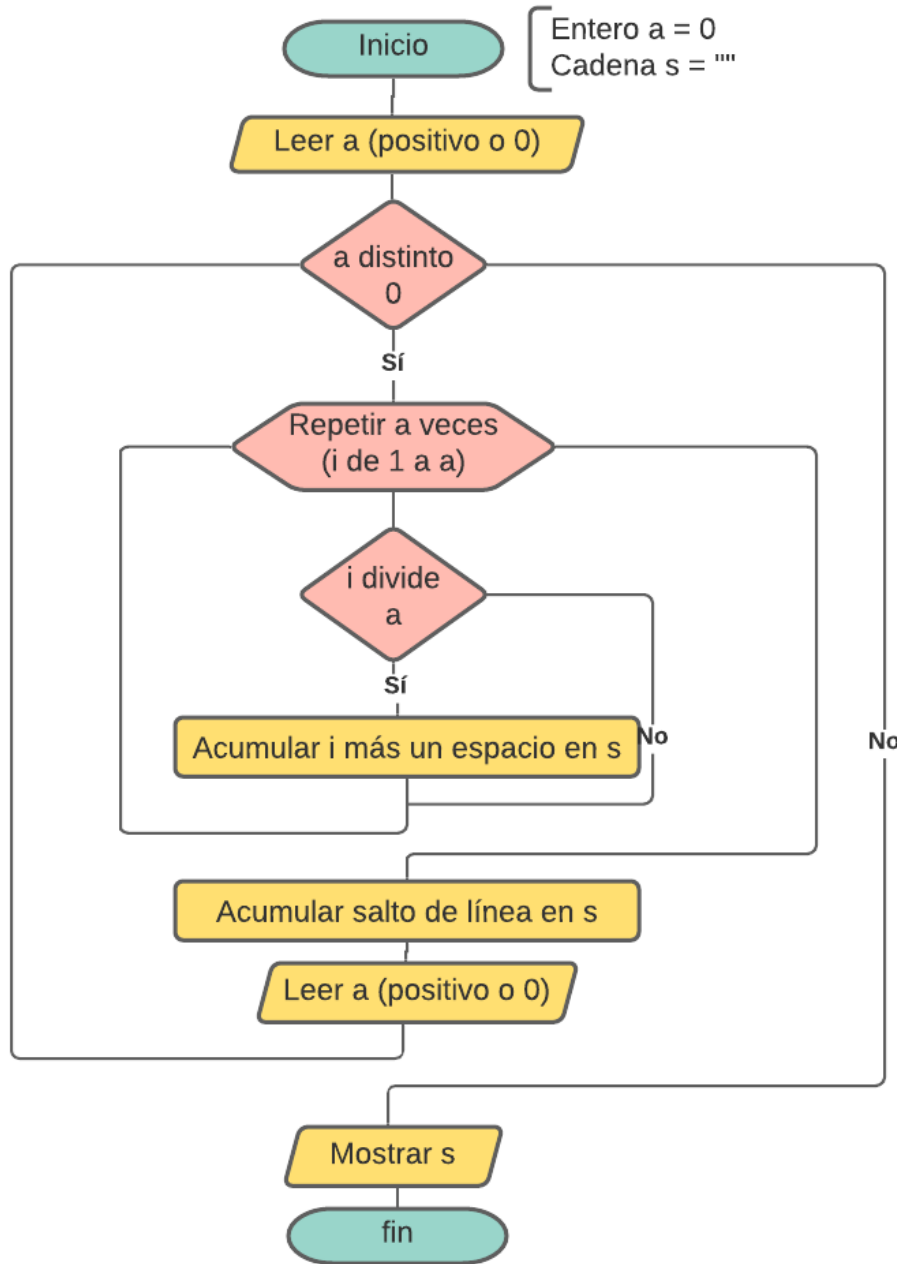
Condición para

Para "i" de 1 a "a"

Algoritmo DIVISORES DE "a"

1. Para "i" de 1 a "a"
 1. Si "i" divide a "a"
 1. Acumular "i" y un espacio en "s"
2. Acumular "salto de línea" en "s"

Anidamiento de bucles (II)



//Variables

```
int a = 0;  
String s = "";
```

Salida
consola

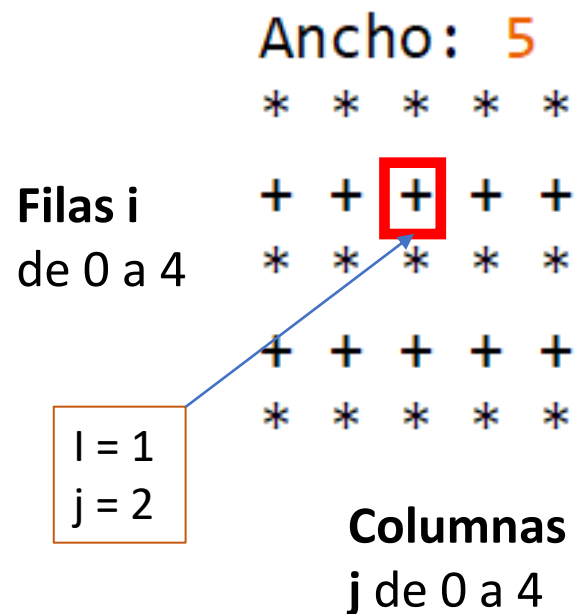
```
Entero (positivo o 0): 5  
Entero (positivo o 0): 12  
Entero (positivo o 0): 0  
1 5  
1 2 3 4 6 12
```

//Programa

```
a = in.leerInt("Entero (positivo o 0): "  
, v->v>=0);  
while (a!=0)  
{  
    for(int i=1;i<=a;i++)  
        if(a%i==0) s+=i+" ";  
  
    s+="\n";  
    a = in.leerInt("Entero (positivo o 0): "  
, v->v>=0);  
}  
System.out.println(s);
```

Anidamiento de bucles (for anidados)

Hágase un programa que lea el ancho (impar) de un cuadrado y muestra un tablero de ancho x ancho cuyas filas pares tienen un * y el resto +.



Datos

Entero ancho = 0

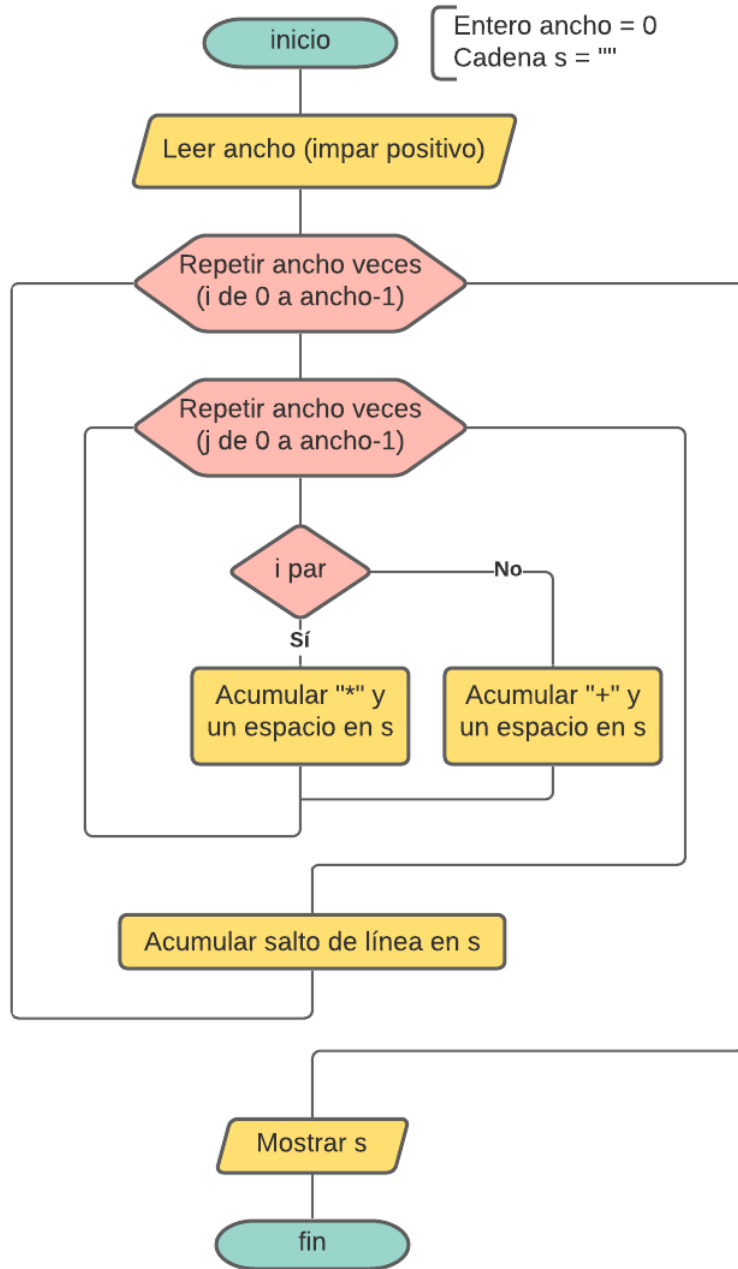
Salida s = ""

Algoritmo

1. Leer "ancho" (positivo impar)
2. Para "i" de 0 a "ancho"-1
 1. RELLENAR FILA "i"
 2. Acumular salto de línea en s
3. Mostrar "s"

Algoritmo RELLENAR FILA "i"

1. Para "j" de 0 a "ancho" - 1
 1. Si "i" es par
 1. Acumular "*" y un espacio en "s"
 2. Si no
 1. Acumular "+" y un espacio en "s"



Anidamiento de bucles (for anidados)

//Variables

```
int ancho = 0;
String s = "";
```

//Programa

```
ancho = in.leerInt("Ancho: ", v->v%2==1
&& v>0);
```

```
for(int i=0;i<ancho;i++)
{
    for(int j=0;j<ancho;j++)
        if(i%2==0) s+="* ";
        else s+="+ ";
    s+="\n";
}
System.out.println(s);
```

Salida
consola

Ancho: 5

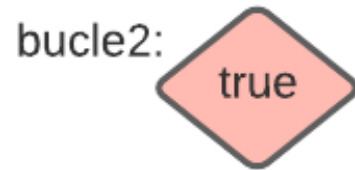
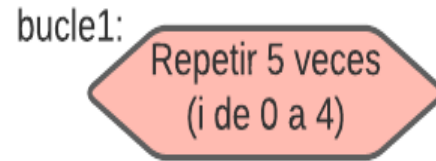
```
* * * * *
+ + + + +
* * * * *
+ + + + +
* * * * *
```



Etiquetas y salto “break”

Con la instrucción “break” es posible abandonar cualquier bucle que contenga dicha instrucción.

Las etiquetas son identificadores que terminan en ‘:’



```
//Variables
int a = 0, b = 0, c = 0;
//Programa
bucle1: for(int i = 0; i <5; i++)
{
    a = (int) (Math.random()*10);
    bucle2: while(true)
    {
        b = (int) (Math.random()*10);
        if(a == b) break bucle1;
        for(int j=0;j<5;j++)
        {
            c = (int) (Math.random()*10);
            if(b==c) break bucle2;
        }
        System.out.println(""+a+b+c);
    }
    System.out.println(""+a+b+c);
}
```

Salida consola →

044
744
277
922
006

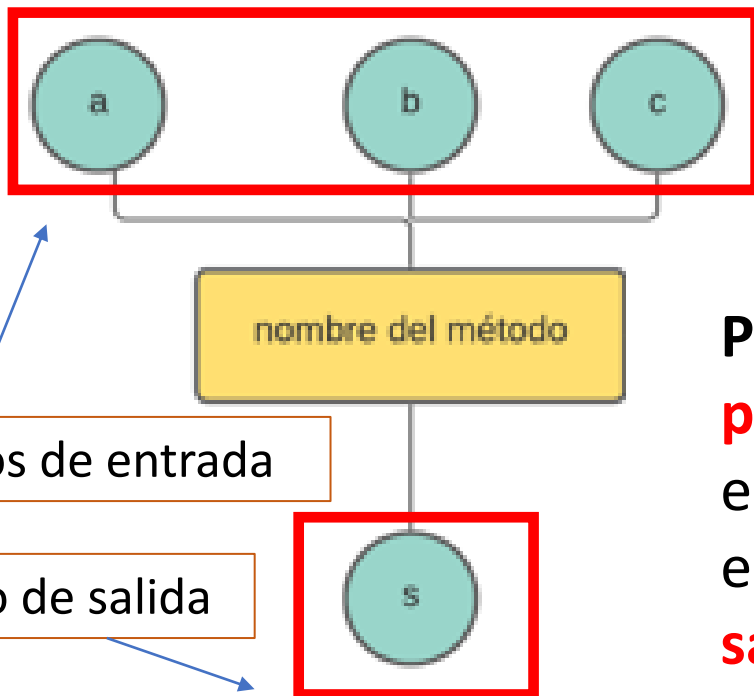
Funciones y procedimientos

Objetivos

- 1) Características de un método: Firma y parámetros
- 2) Funciones
- 3) Procedimientos
- 4) Métodos recursivos
- 5) Menús

Métodos en programación estructurada

Un método en programación estructurada es una subrutina que tiene un nombre y que realiza una tarea para unos datos de entrada y puede obtener un dato de salida y modificar datos que se pasan a la subrutina



Funciones Son métodos que tienen datos de salida

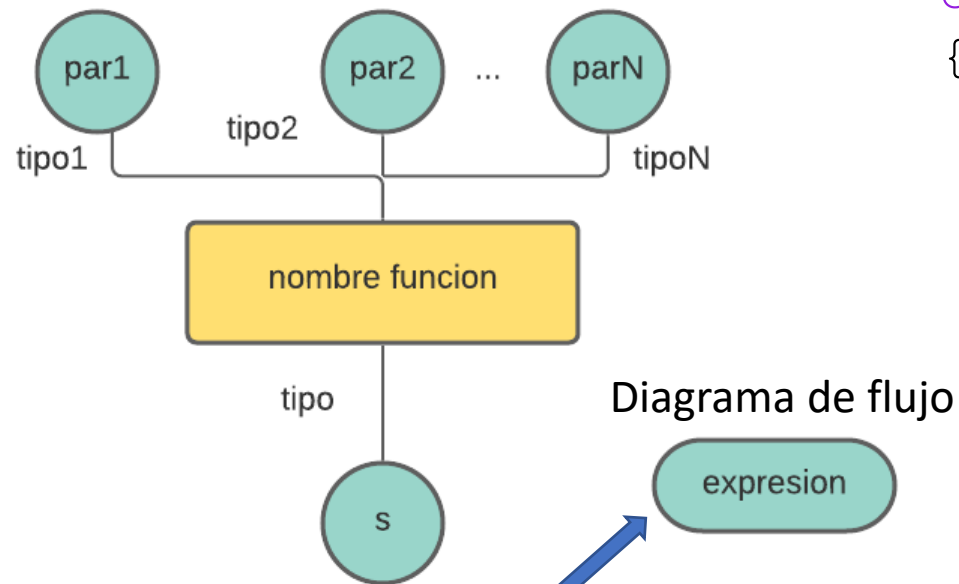
Procedimientos Son métodos que no tienen datos de salida

Parámetros A los datos de entrada se denominan **parámetros por valor** si no se modifican los datos de entrada y **por referencia** si se modifican lo dato de entrada. Al dato de salida se denomina **parámetro de salida**

Definición de funciones

static <tipo> <nombre-función> (<tipo1> par1 , ... , <tipoN> parN)

bloque



La instrucción **return expresión;** termina la ejecución de la función devolviendo el valor de la expresión. Se puede poner varias veces.

```
class EjemploFuncion
```

```
{
```

```
public static void main(String[] args)
```

```
{
```

Tipo del parámetro de salida

nombre

Parámetros por valor

```
static int mcm(int a, int b)
```

Firma

```
{
```

```
//Variables  
int v = 0;
```

Datos Incluye una variable de tipo el parámetro de salida

```
//Programa
```

```
return v ;
```

Algoritmo Consiste en calcular "v"
1. Devolver "v"

Puede ser una expresión de tipo el parámetro de salida

```
}
```

```
}
```


Funciones

Defínase una función que dado dos enteros (parámetros) devuelva la suma de los dos

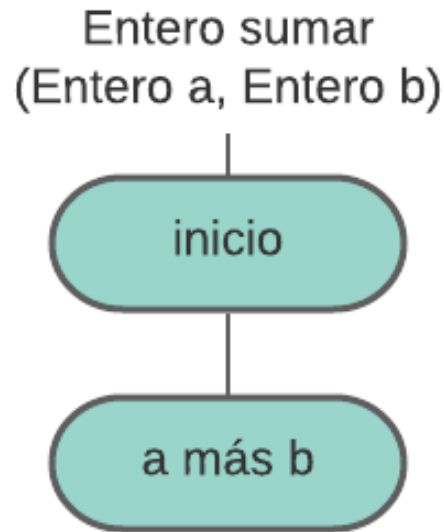
Firma

static int sumar(int a, int b)

Datos

Algoritmo

1. Devolver la suma de "a" y "b"



```
class EjemploSumar
{
    public static void main(String[] args)
    {
    }

    static int sumar(int a, int b)
    {
        //Variables

        //Programa
        return a + b;
    }
}
```

En la definición de una función sólo se pondrá a partir de ahora su **firma**, sus **datos** y el **algoritmo**

Invocar una función

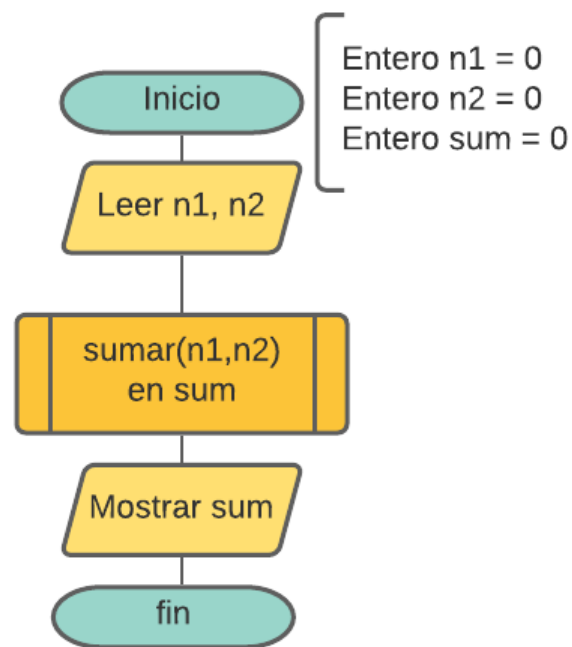
Hágase un programa que lea dos enteros y obtenga su suma

Datos

int n1 = 0
int n2 = 0
int sum = 0

Algoritmo

1. Leer "n1"
2. Leer "n2"
3. sumar(n1,n2) en "sum"
4. Mostrar "sum"



```
import fsg.in;  
class EjemploSumar  
{
```

PROGRAMA COMPLETO

```
public static void main(String[] args)  
{
```

```
//Variables
```

```
int n1 = 0, n2 = 0, sum = 0;
```

```
//Programa
```

```
n2 = in.leerInt();
```

```
sum = sumar(n1,n2);  
System.out.println(sum);
```

```
}
```

```
static int sumar(int a, int b)
```

```
{
```

```
//Variables
```

```
//Programa
```

```
return a + b;
```

```
}
```

```
}
```

A "n1" y "n2" se denominan argumentos (son del mismo tipo que los parámetros) y se copian en los parámetros

Parámetros por valor

El valor de "n1" se copia en "a"
El valor de "n2" se copia en "b"
Se obtiene a+b y se copia en "sum"

Tabla de ejecución con funciones

Hágase una tabla de ejecución del programa anterior con la entrada de 5 y 7

Ins(Con	Con	n1	n2	sum
Inicializacion		0	0	0
n1 = in.leerInt();		5	0	0
n2 = in.leerInt();		5	7	0
sum = sumar(n1,n2)		5	7	12
System.out.println(sum)	12	5	7	12

Se calculan los argumentos de la función y se hace una tabla de ejecución para la función con tales argumentos

sumar(5,7)

Ins/Con	Con	a	b
Parámetros		5	7
return a+b	12	5	7

En la columna Consola/Condición se indica el valor devuelto en rojo

Como la función no tiene datos no aparece la fila de Inicialización de los datos. Se pondría después de la de "Parámetros"



Argumentos de una función

Como argumentos de una función se puede poner cualquier expresión del tipo el parámetro en cuestión

Por ejemplo, para la función de firma **static int sumar(int a, int b)** se tienen los siguientes ejemplo de paso de argumentos:

```
//sum, n1=4, n2=3 int  
//d=5.7 double
```

```
sum = sumar(2, 8);
```

Como argumentos se pasan los valores 2 y 8

```
sum = sumar(n1, 1);
```

Como argumentos se pasan los valores 4 (n1) y 1

```
sum = sumar(2*n2+5, (int) d)
```

Como argumentos se pasan los valores 11 ($2*3+5$) y 5 ((int)5.7)

```
sum = sumar((int) (Math.random()*10), (int) (Math.random()*10));
```

Como argumentos se pasan dos números aleatorios entre 0 y 9

Uso de una función

El resultado devuelto por una función se puede utilizar en cualquier expresión que admita el tipo del parámetro de salida

Por ejemplo, para la función de firma **static int sumar(int a, int b)** se tienen los siguientes ejemplos de uso de la función:

En una instrucción de asignación `sum = sumar(2, n2+1);`

`sum = 3*sumar(2, n2+1) - n2;`

En las evaluaciones de las expresiones
las funciones tienen máxima prioridad

En una instrucción de salida

`System.out.println("La suma es: ", sum(a, b));`

En una instrucción condicional `if (sumar(2, a) > 10) ...`

En una instrucción de devolución `return sumar(a+1, 10) * 2;`

Ejemplo función

Hágase una función que dado un entero (parámetro) obtenga el número de dígitos

Firma

static int digitos(int n)

Estrategia

Quitar las unidades de "n" hasta llegar a 0

Datos

int con = 0

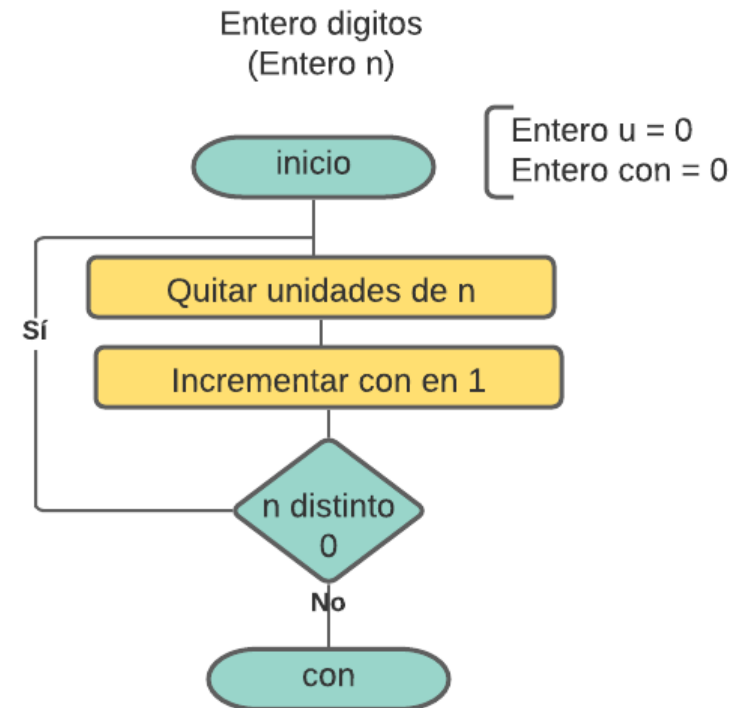
Condición mientras

Si "n" distinto de 0

Algoritmo

1. Quitar las unidades de "n"
2. Incrementar "con" en 1
3. Si "n" distinto de 0
4. Devolver "con"

Almacena el número de dígitos que tiene "n"



Ejemplo función (a)

```
static int digitos(int n)
{
    //Variables
    int con = 0;

    //Programa
    do
    {
        n = n /10;
        con++;
    }while (n!=0) ;

    return con;
}
```

Tabla de ejecución para digitos(230)

Ins/Con	Con	n	con
Parámetros		230	
Inicialización		230	0
n = n /10		23	0
con++		23	1
n!=0	false	23	1
n = n /10		2	0
con++		2	2
n!=0	false	2	2
n = n /10		0	2
con++		0	3
n!=0	true	0	3
return con	3		



Ejemplo funciones (b)

Hágase un programa que lea tres enteros y muestre el número de dígitos que tiene cada dato leído

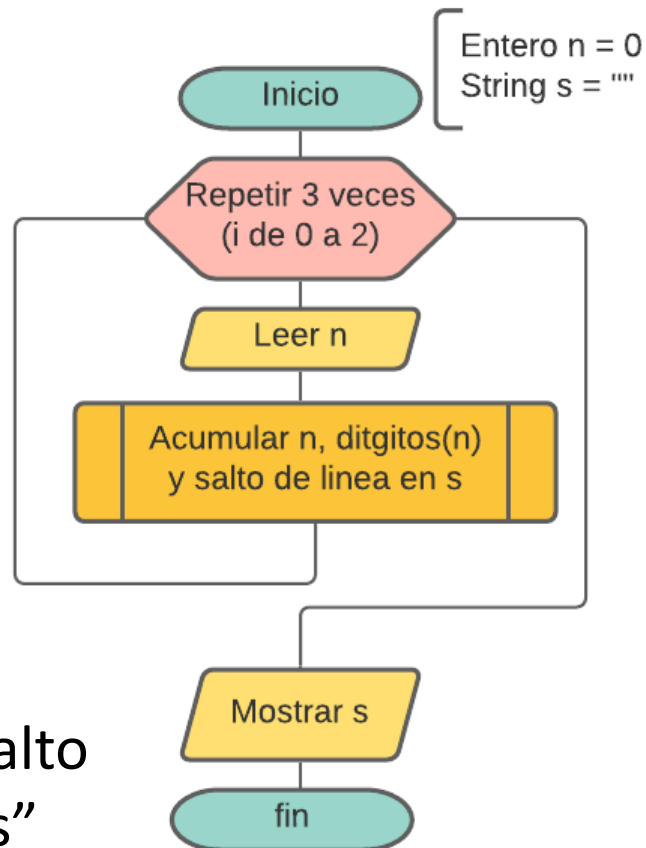
Datos

int n = 0
String s = ""

Condición para
Para "i" de 0 a 2

Algoritmo

1. Para "i" de 0 a 2
 1. Leer "n"
 2. Acumlar "n", dígitos(n) y salto de linea en "s"
2. Mostrar "s"



//Variables

```
int n = 0;  
String s = "";
```

//Programa

```
for(int i=0; i<3; i++)  
{  
    n = in.leerInt();  
    s+=n+" tiene " +  
        digitos(n) + " digitos\n";  
}
```

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

ESCRIBE UN ENTERO: 7
ESCRIBE UN ENTERO: 5672
ESCRIBE UN ENTERO: 300
7 tiene 1 digitos
5672 tiene 4 digitos
300 tiene 3 digitos

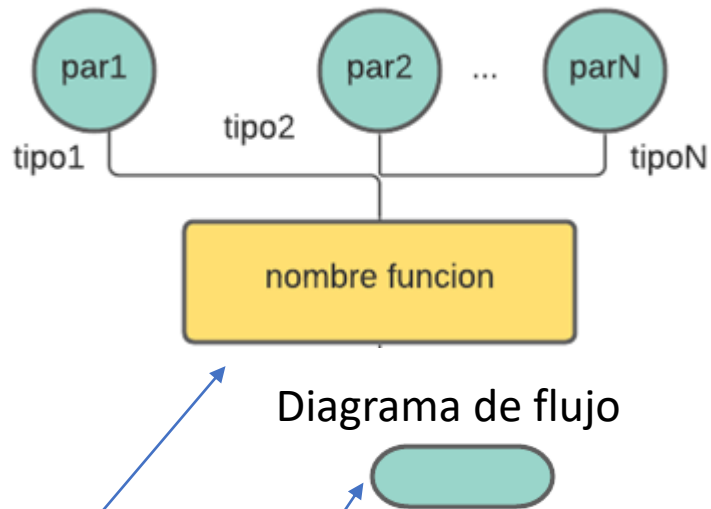
Salida
consola



Procedimientos con parámetros por valor

static **void** <nombre-función> (<tipo1> par1 , ... , <tipoN> parN)

bloque



La instrucción **return**; termina la ejecución del procedimiento. Se puede poner varias veces.

```
import fsg.in;
class EjemploProcedimiento
{
    public static void main(String[] args)
    {
        int a = 0;
        a = in.leerInt();
        mostrar("EL valor leído es ", a);
    }
    static void mostrar(String mensaje, int n)
    {
        //Variables
        //Programa
        System.out.println(mensaje + n);
    }
}
```

PROGRAMA COMPLETO

Esta es la única forma de invocar un procedimiento

mostrar("EL valor leído es ", a);

//Variables
//Programa

Datos

Firma

Algoritmo

System.out.println(mensaje + n);

Ejemplo procedimiento

Hágase un procedimiento que dado dos enteros (parámetros ancho y alto) muestre un rectángulo de esas características lleno de asteriscos.

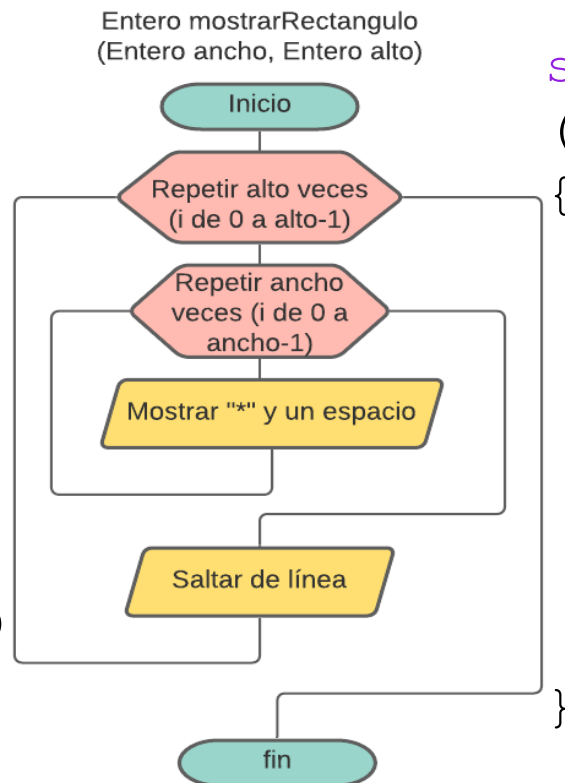
Firma

static void mostrarRectangulo
(int ancho, int alto)

Datos

Algoritmo

1. Para "i" de 0 a alto-1
 1. Para "j" de 0 a ancho -1
 1. Mostrar "*" y espacio
 2. Saltar de linea



```
static void mostrarRectangulo  
(int ancho, int alto)  
{  
    for(int i=0; i<alto; i++)  
    {  
        for(int j=0; j<ancho; j++)  
            System.out.print("* ");  
  
        System.out.println();  
    }  
}
```



Tipos básicos y de cadena por referencia

- En java no se puede pasar datos básicos o de cadena por referencia
- Se puede simular con una función que se le pasa el dato a modificar (nombre de la variable) y se devuelve el valor modificado

```
static String concatenar(String c,String s)
{
    c = c+s;
    return c;
}
```

Función que concatena a una cadena otra cadena

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

Función que duplica un valor entero

Inconveniente: No se pueden modificar dos datos de entrada en una misma función

//Variables

```
int n = 3;
String cad = "Hola";
```

6
Hola Adios

//Programa

```
cad = concatenar(cad, " Adios");
```

```
n = duplicar(n);
```

```
System.out.println(n);
System.out.println(cad);
```

Salida
consola



Métodos recursivos

Un método recursivo es un método que se llama a sí mismo

Hágase un método que obtenga el factorial de un número según la formula siguiente: $0! = 1$; $n! = (n-1)! * n$ si $n > 0$

Firma

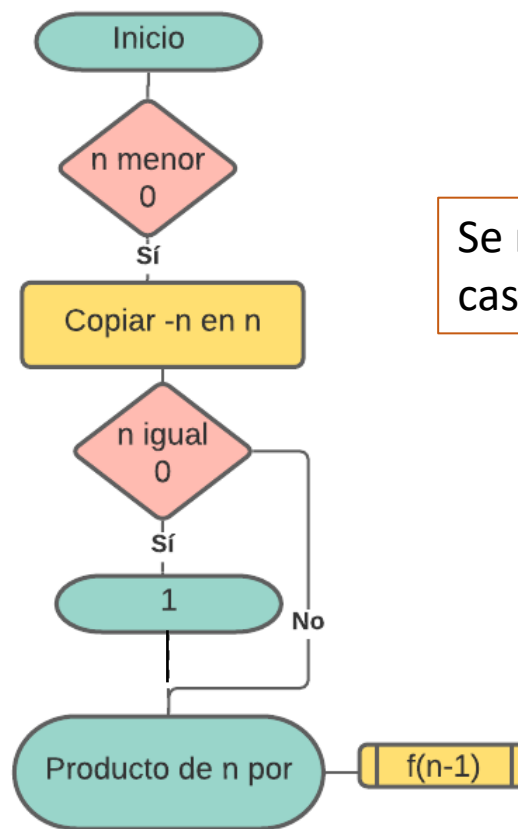
static int f(int n)

Datos

Algoritmo

1. Si "n" < 0
 1. Copiar -n en "n"
2. Si "n" igual 0
 1. Devolver 1
3. Devolver $n * f(n-1)$

Entero f(Entero n)



Se resuelve para casos concretos

```
static int f(int n)
{
    if (n < 0) n = -n;
    if (n == 0) return 1;

    return n * f(n-1);
}
```

Tabla de ejecución de método recursivo

Tabla de ejecución del método anterior para $f(3)$

Ins/Con	Con	n	Ins/Con	Con	n
Parámetros		3	Parámetros		1
$n < 0$	false	3	$n < 0$	false	1
$n == 0$	false	3	$n == 0$	false	1
return $n * f(n-1)$	$3 * f(2)$	3	return $n * f(n-1)$	$1 * f(0)$	1
Parámetros		2	Parámetros		0
$n < 0$	false	2	$n < 0$	false	0
$n == 0$	false	2	$n == 0$	true	0
return $n * f(n-1)$	$2 * f(1)$	2	return 1	1	0



Tabla de ejecución de método recursivo (a)

Ins/Con	Con	n	Ins/Con	Con	n
Parámetros		3	Parámetros		1
n<0	false	3	n<0	false	1
n==0	false	3	n==0	false	1
return n*f(n-1)	3*f(2)	3	return n*f(n-1)	1	1
Parámetros		2			
n<0	false	2			
n==0	false	2			
return n*f(n-1)	2*f(1)	2			




Tabla de ejecución de método recursivo (b)

Ins/Con	Con	n
Parámetros		3
$n < 0$	false	3
$n == 0$	false	3
return $n * f(n-1)$	$3 * f(2)$	3
Parámetros		2
$n < 0$	false	2
$n == 0$	false	2
return $n * f(n-1)$	2	2



Ins/Con	Con	n
Parámetros		3
$n < 0$	false	3
$n == 0$	false	3
return $n * f(n-1)$	6	3



```
//Variables
```

```
int opcion = 0;
```

Menús

```
//Programa
```

```
do  
{
```

```
    in.cls();
```

```
    System.out.println("OPCION ACCION\n"+
```

```
        "===== \n"+
```

```
        "    1    Opcion 1\n"+
```

```
        "    2    Opcion 2\n"+
```

```
        " otra   Terminar");
```

Se pueden añadir opciones

```
    opcion = in.leerInt("OPCION: ");
```

```
    in.cls();
```

```
    switch(opcion)
```

```
    {
```

```
        case 1: in.detener(); break;
```

```
        case 2: in.detener(); break;
```

```
    }
```

```
} while(1<=opcion && opcion<=2);
```

OPCION ACCION

=====

1 Opcion 1

2 Opcion 2

otra Terminar

OPCION:

Cada opción tendrá un texto más descriptivo

Para cada opción se define un procedimiento del siguiente tipo:

```
static void opcion1()
```

```
{
```

```
    in.detener();
```

```
}
```

Se deben poner nombres más descriptivos a los métodos

Se sustituye por
opcion1()

Se sustituye por
opcion2()



Menús (II)

Programa que muestra un menú con dos opciones en el que se ha ejecutado previamente la primera

```
//Variables  
int opcion = 1;
```

```
//Programa  
do  
{
```

```
    in.cls();  
    switch(opcion)  
    {
```

```
        case 1: System.out.println("Opcion 1"); break;
```

```
        case 2: System.out.println("Opcion 2"); break;
```

```
    }
```

```
    System.out.println("1. Opcion1 2. Opcion2");
```

```
    opcion = in.leerInt("OPCION: ");
```

```
} while(1<=opcion && opcion<=2);
```

Para cada opción se define un procedimiento del siguiente tipo:

```
static void opcion1()  
{  
    System.out.println("Opción 1");  
}
```

Se deben poner nombres más descriptivos a los métodos

Se sustituye por `opcion1()`

Se sustituye por `opcion2()`

Se pueden añadir opciones

Cada opción tendrá un texto más descriptivo

```
Opcion 1  
1. Opcion1 2. Opcion2  
OPCION: █
```

Instrucciones de error

Objetivos

- 1) Instrucciones de error
- 2) Crear un objeto de excepción
- 3) Instrucción throw
- 4) Instrucción try-catch

Instrucciones de error

En java se define dos instrucciones para manejo de excepciones:

1. Lanzar una excepción
2. Capturar una excepción.

En java se tienen los siguientes tipos de excepciones (observa el seleccionado)

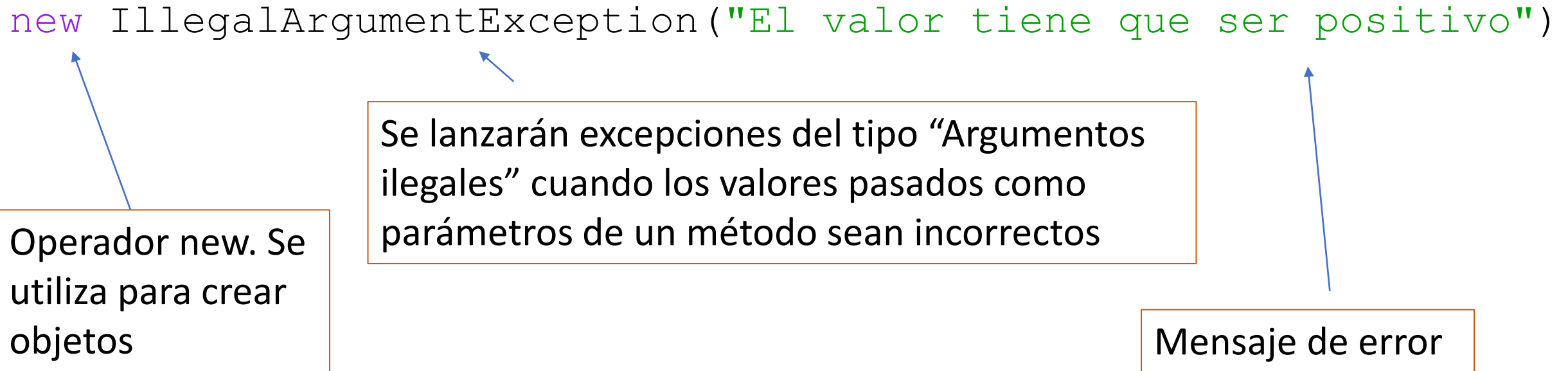
AnnotationTypeMismatchException, ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, CatalogException, ClassCastException, ClassNotPreparedException, CMMException, CompletionException, ConcurrentModificationException, DateTimeException, DOMException, DuplicateRequestException, EmptyStackException, EnumConstantNotPresentException, EventException, FileSystemAlreadyExistsException, FileSystemNotFoundException, FindException, **IllegalArgumentException**, IllegalCallerException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, IllformedLocaleException, ImagingOpException, InaccessibleObjectException, IncompleteAnnotationException, InconsistentDebugInfoException, IndexOutOfBoundsException, InternalException, InvalidCodeIndexException, InvalidLineNumberException, InvalidModuleDescriptorException, InvalidModuleException, InvalidRequestStateException, InvalidStackFrameException, JarSignerException, JMRuntimeException, JSEException, LayerInstantiationException, LSEException, MalformedParameterizedTypeException, MalformedParametersException, MirroredTypesException, MissingResourceException, NativeMethodException, NegativeArraySizeException, NoSuchDynamicMethodException, NoSuchElementException, NoSuchMechanismException, NullPointerException, ObjectCollectedException, ProfileDataException, ProviderException, ProviderNotFoundException, RangeException, RasterFormatException, RejectedExecutionException, ResolutionException, SecurityException, SPIResolutionException, TypeNotPresentException, UncheckedIOException, UndeclaredThrowableException, UnknownEntityException, UnmodifiableModuleException, UnmodifiableSetException, UnsupportedOperationException, VMDisconnectedException, VMMismatchException, VMOutOfMemoryException, WrongMethodTypeException, XPathException

Crear un objeto de excepción

Para crear un objeto de tipo una excepción del tipo “`IllegalArgumentException`” se utiliza el siguiente código:

```
new IllegalArgumentException("El valor tiene que ser positivo")
```

Operador new. Se utiliza para crear objetos



Se lanzarán excepciones del tipo “Argumentos ilegales” cuando los valores pasados como parámetros de un método sean incorrectos

Mensaje de error

Instrucción throw

La instrucción throw se utiliza para lanzar una excepción

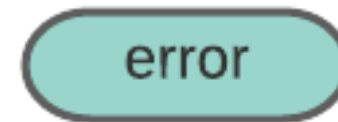
```
throw new IllegalArgumentException("Valor negativo");
```

Al ejecutarse dicha instrucción se termina la ejecución del método lanzando una excepción y devolviendo el control a la **sentencia** donde se invoco dicho método. Para dicha sentencia, como se verá, se puede capturar y gestionar el error para no terminar abortando la aplicación

Algoritmo

1. Lanzar error

Diagrama de flujo



Ejemplo de instrucción throw

Hágase una función que dados dos enteros positivos (parámetros) se obtendrá su suma.

Firma

static int sumar(int a, int b)

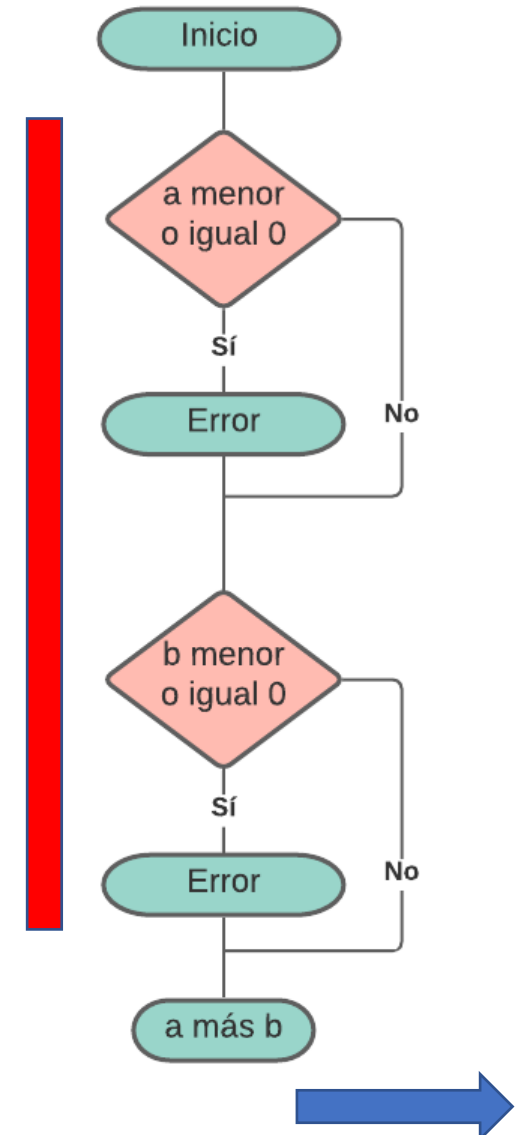
Datos

Algoritmo

1. Si "a" menor o igual que 0
 1. Lanzar error
2. Si "b" menor o igual que 0
 1. Lanzar error
3. Devolver la suma de "a" y "b"

Validación de parámetros por valor

Entero sumar
(Entero a, Entero b)



Ejemplo de instrucción throw (a)

```
static int sumar(int a, int b)
{
    if (a<=0)
        throw new IllegalArgumentException("Dato no positivo");

    if (b<=0)
        throw new IllegalArgumentException("Dato no positivo");

    return a+b;
}
```

Se deberían validar los parámetros de un método antes de definir las variables y el programa de dicho método

Instrucción try-catch

Se puede capturar y gestionar excepciones con la instrucción try-catch

Tipo de excepción a capturar

Excepción genérica

```
try
{
    //Instrucciones
}
catch (IllegalArgumentException e)
{
    //Gestionar error IllegalArgumentException
}
catch (Exception e)
{
    //Gestionar error Exception
}
finally
{
}
```

La primera instrucción que provoque una excepción, cede el control al primer “catch” que capture dicha excepción o al último (“Exception”) si no hubiera ninguno. Por último se ejecuta las instrucciones de “finally” (es opcional)

Variable para acceder a la excepción

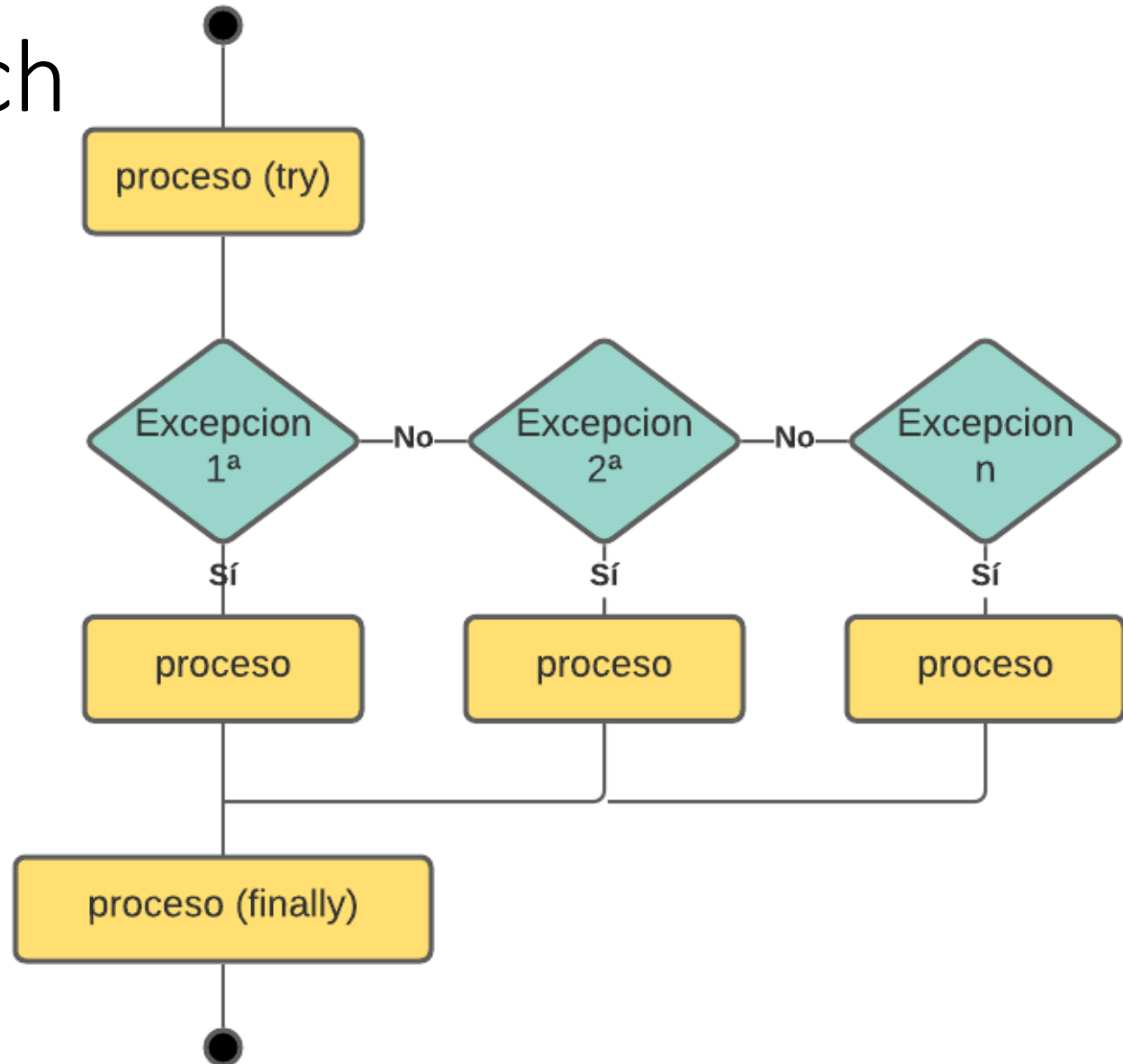
SUGERENCIA Si se quiere mostrar la traza de la excepción, se puede escribir en la clausula catch la instrucción: **e.printStackTrace();**



Instrucción try-catch

Algoritmo

1. • Instrucciones try
2. Si excepción 1
 1. Instrucciones
3. Si no si excepción2
 1. Instrucciones
4. ...
5. Instrucciones finally •



Ejemplo instrucción try-catch

Hágase un programa que lea 5 pares de enteros. Para cada pareja de datos se sumarán utilizando el último método de sumar definido. Se capturarán los posibles errores que puede lanzar el método. De no lanzarse error, se mostrará el resultado de la suma; si no, se mostrará un mensaje diciendo que “No se puede sumar”

```
//Variables
```

```
int a = 0, b = 0;
```

```
int sum = 0;
```

```
//Programa
```

```
for(int i=0;i<5;i++)
```

```
{
```

```
    a = in.leerInt();
```

```
    b = in.leerInt();
```

```
    try
```

```
    {
```

```
        sum = sumar(a,b);
```

```
        System.out.println(" "+a+" "+" "+b+" "+"=" +sum);
```

```
    }
```

```
    catch (Exception e)
```

```
    {
```

```
        System.out.println("No se puede sumar");
```

```
    }
```

```
}
```



```
ESCRIBE UN ENTERO: 5
```

```
ESCRIBE UN ENTERO: 7
```

```
5+7=12
```

```
ESCRIBE UN ENTERO: 2
```

```
ESCRIBE UN ENTERO: -6
```

```
No se puede sumar
```

```
ESCRIBE UN ENTERO: 0
```

```
ESCRIBE UN ENTERO: 9
```

```
No se puede sumar
```

```
ESCRIBE UN ENTERO: 8
```

```
ESCRIBE UN ENTERO: 2
```

```
8+2=10
```



Librerías de usuario

Objetivos

- 1) Saber qué es una librería de usuario
- 2) Definir funciones y procedimientos en una librería
- 3) Uso de métodos de una librería en una aplicación
- 4) Sobrecarga de métodos

Librerías de usuario

Una librería de usuario es una clase definida en un directorio y que contienen funciones y procedimientos. Tendrá las siguientes características:

- Se definirá en un paquete (o directorio)
- El nombre de la clase será en minúscula
- La clase será pública
- Las funciones serán públicas
- Los procedimientos serán públicos

Ejemplos de firmas de métodos definidos en “lib”:

```
public static int sumar(int a, int b)
static public void mostrar(double v)
```



```
package fsg;
public class lib
{
}
```



 lib.class

 lib.java

El directorio “fsg”
estará definido
donde está la
aplicación con el
“main”

Métodos en una librería

Defínase en la librería “lib” un método para sumar dos enteros.

```
package fsg;  
public class lib  
{  
    public static int sumar(int a, int b)  
    {  
        return a+b;  
    }  
}
```

Uso de librerías en aplicaciones

Para utilizar los métodos definidos en la librería “lib” en una aplicación se hace del siguiente modo:

- Se importa la librería
- Se invoca un método de la librería anteponiendo al nombre del método el nombre de la librería, en este caso “lib”, seguido de un punto

```
import fsg.lib ;
```

```
import fsg.in;
```

```
class EjemploAccesoLibrería
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
    //Variables
```

```
    int a = 0, b = 0, suma = 0;
```

```
    //Programa
```

```
    a = in.leerInt();
```

```
    b = in.leerInt();
```

```
    suma = lib.sumar(a,b) ;
```

```
    System.out.println("Suma: "+suma);
```

```
}
```

```
}
```

```
//Importar todas las clase del paquete “fsg”
```

```
import fsg.*;
```

fsg

EjemploAccesoLibrería.class

EjemploAccesoLibrería.java

Sobrecarga de métodos en librerías de usuario

Un método se dice que está sobrecargado en una clase si en dicha clase existe otro método con el mismo nombre pero con distinto número de parámetros o de distinto tipo.

Ejemplo de firmas de métodos sobrecargados serían:

```
public static int sumar(int a, int b)  
public static double sumar(int a, double b)  
public static double sumar(double a, int b)  
public static int sumar(int a, int b, int c)  
public static double sumar(double a, double b)
```

En tiempo de ejecución se ejecuta un método u otro en función del tipo de los argumentos

¿Cuál de los métodos “sumar” se ejecutaría para la sentencia `sum = sumar(45,23L);` ?



La clase String

Objetivos

- 1) Saber definir expresiones regulares. Método matches
- 2) Validar datos de entrada de tipo cadena
- 3) La clase String
- 4) La clase StringBuffer

Expresiones regulares

Las expresiones regulares son cadenas que representan patrones.

- Con las expresiones regulares se puede comprobar si una cadena tiene o no un determinado formato.
- Se puede verificar si una cadena tiene un formato numérico, es un número de teléfono, un dni, una palabra, una cuenta bancaria, un email, una página web, etc
- Una expresión regular se forma a partir de una secuencia de caracteres del tipo el mostrado en las tablas siguientes:

Codificación de caracteres

Expresión	Encaja con
x	El carácter x
\\uhhhh	El carácter con valor hexadecimal 0x hhhhh
\\t	El tabulador ('u0009')
\\n	Nueva línea ('u000A')
\\r	Retorno de carro ('u000D')
\\f	Nueva página ('u000C')
\\a	Un beep de alerta ('u0007')
\\e	Escape ('u001B')

Intervalos de caracteres

Expresión	Encaja con
[abc]	a, b o c
[^abc]	Cualquier carácter menos a, b o c (negación)
[a-zA-Z]	Desde la a a la z o desde la A a la Z , incluidos
[x[y][z]]	Unión de los intervalos de caracteres x, y y z
[x&&y&&z]	Intersección de los intervalos de caracteres x, y y z

Ejemplos:

[a-z&&[^aeiou]] → Todas las letras de la 'a' a la 'z' salvo las vocales

[a-zA-ZñÑáéíóúüÁÉÍÓÚÜ] → Cualquier letra del abecedario español

Intervalos de caracteres predefinidos

Expresión	Encaja con
.	Cualquier carácter
\\d	Un número: [0-9]
\\D	Todo menos un número: [^0-9]
\\s	Un espacio en blanco: [\\x0Bf]
\\S	Todo menos un espacio en blanco: [^\\s]
\\w	Una letra: [a-zA-z]
\\W	Todo menos una letra: [^a-zA-z]

Cuantificadores de cantidad

Expresión	Encaja con
$X?$	X , una o ninguna vez
X^*	X , cero o muchas veces
X^+	X , una o más veces
$X\{n\}$	X , exactamente n veces
$X\{n,m\}$	X , por lo menos n veces pero no más de m veces

Ejemplos:

$[0-9]^+$ → Al menos un dígito.

$[aeiou]\{3,5\}$ → Como mínimo 3 vocales y como máximo 5

$[0-9]\{9\}$ → Nueve dígitos (números del 00000000 al 99999999)

$.*[aeiou]$ → Cadenas que terminan en vocal

Operadores lógicos

Expresión	Encaja con
XY	X seguido de Y
X Y	X o Y
(X)	X como un grupo

Ejemplos:

[0-9]{8}[A-Z] → Ocho dígitos seguidos de una letra.

[A-ZÑ][a-zñ]* → Palabra que comienza por mayúsculas y el resto en minúsculas

0|100|[1-9][0-9]? → Números del 0 al 100

[Ll]unes|[Mm]artes|[Mm]iercorles|[Jj]ueves|[Vv]iernes → Día de la semana L-V

(SI|NO)? → SI, NO o la cadena vacía

[a-zñáéíóúü]+(\\s[a-zñáéíóúü]+)? → Palabras en minúsculas separadas por un espacio

Caracteres especiales con escape

Exp.	Encaja con
\\.	El punto (.)
\\	La barra (\)
*	El asterisco (*)
\\+	El mas (+)
\\?	El interrogante cerrado (?)
\\[El corchete abierto ([)
\\]	El corchete cerrado (])

Exp.	Encaja con
\\{	La llave abierta ({)
\\}	La llave cerrada (})
\\	La tubería ()
\\,	La coma (,)
\\-	El guión (-)
\\&	El ampersand (&)
\\^	El circunflejo (^)

Ejemplo:

0\\. [0-9]{1,2} → Números reales entre 0.00 y 0.99 con al menos 1 decimal

Ejemplos de expresiones regulares

Que aparezcan exactamente dos puntos `[^\\.]*\\.([^\\.]*\\.([^\\.]*)`

Que aparezcan al menos dos vocales juntas `.*[aeiouáéíóúüAEIOUÁÉÍÓÚÜ]{2,}.*`

Natural par entre 4 y 100 `[1-9][02468]|100|[468]`

Que comienza por consonante en mayúsculas `[B-ZÑ&&[^EIOUÁÉÍÓÚ].*`

Número real con valores entre 10.00 y 50.00 (los decimales no pueden terminar en 0, ni tampoco poner punto decimal sin decimales válidos)

`[1-4][0-9](\\.([1-9])|\\.([0-9][1-9]))?|50`

El método “matches” de las cadenas

El método “matches” de las cadenas comprueba si una cadena cumple o no una expresión regular.

//String s

s.matches ("Uno | Dos | Tres")

Expresión regular

Devuelve verdadero si “s” es la cadena “Uno”, “Dos” o “Tres”; si no, falso

s.matches (" [0-9] {24} ")

Devuelve verdadero si “s” es una cadena de 24 dígitos; si no, falso

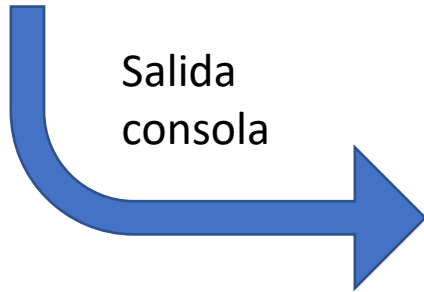
s.matches (" [A-ZÑ] [a-zñ] * ")

Devuelve verdadero si “s” es una palabra con la primera letra en mayúsculas y el resto en minúsculas

Validación de datos de entrada de cadena

Se pueden utilizar los métodos “leerString” y “leerLine” de la librería “in” para validar la cadena introducida de forma que cumpla una determinada expresión regular

```
// "palabra" String  
palabra = in.leerString("Escribe una palabra que  
termine en vocal: ", v->v.matches(".*[aeiou]"));
```



Salida
consola

```
Escribe una palabra que termine en vocal: Color  
Valor incorrecto  
Escribe una palabra que termine en vocal: Hola
```

Validación de datos de entrada de cadena (II)

Hágase un programa en el que se lea el nombre y los apellidos de una persona (serán palabras separadas por un espacio; cada palabra comienza por mayúscula y el resto en minúsculas)

```
//Variables
```

```
String nombre = "";
```

```
//Programa
```

```
nombre = in.leerLine("Nombre y apellidos: ",  
    v->v.matches("[A-ZÑÁÉÍÓÚ][a-zñáéíóúü]* (\\s[A-ZÑÁÉÍÓÚ][a-zñáéíóúü]*)+"));
```

Nombre y apellidos: LUIS GONZALEZ

Valor incorrecto

Nombre y apellidos: Luis González

Valor incorrecto

Nombre y apellidos: Luis González Argüelles



Salida
consola



La clase String

En Java se tiene una clase “String” definida en el paquete “java.lang”, que permite manejar cadenas.

Creación de una cadena

```
s = "Hola, ¿cómo estás?";  
s1 = new String("Hola");
```

Comparar lexicograficamente la cadena “s” con otra cadena

```
s.compareTo("Hola")  
→ 0 (si son iguales)  
→ <0 si s<"Hola"  
→ >0 si s>"Hola"
```

Obtener de “s” el tercer carácter en “c”

```
c = s.charAt(2);  
c = s[2];
```

Concatenar a la cadena “s” la cadena “s1”

```
s = s.concat(s1);  
s = s + s1;
```

La clase String (II)

Comprobar si la cadena “s” termina en la cadena “s1”

`s.endsWith(s1) → boolean`

Comprobar si la cadena “s” es igual a “s1”

`s.equals(s1) → boolean`

Formatear una cadena

El método format es similar al método “printf” con la diferencia de que el resultado no se muestra por consola

`s = String.format("Día %2d y peso %5f\n", 3, 45.3f);`

Buscar la primera posición que ocupa el carácter ‘a’ en “s” o -1 si no está

`s.indexOf('a') → int`

Buscar la primera posición que ocupa el carácter ‘a’ en “s” desde la posición 3 o -1 si no está

`s.indexOf('a', 3) → int`

La clase String (III)

Comprobar si la cadena “s” está vacía (tiene longitud 0)

`s.isEmpty()` → `boolean`

Buscar la última posición que ocupa el carácter ‘a’ en “s” desde la posición 3 o -1 si no está

`s.lastIndexOf('a', 3)` → `int`

Obtener una cadena reemplazando el carácter ‘a’ por ‘b’ en “s”

`s.replace('a', 'b')` → `String`

Buscar la última posición que ocupa el carácter ‘a’ en “s” o -1 si no está

`s.lastIndexOf('a')` → `int`

Obtener el número de caracteres que tiene “s”

`s.length()` → `int`

Comprobar si “s” cumple una expresión regular

`s.matches("h.*")`

Obtener una cadena reemplazando la cadena “ho” por “HO” en “s”

`s.replace("ho", "HO")` → `String`

La clase String (IV)

Obtener una cadena reemplazando toda subcadena que cumple la expresión regular “a{2,3}” por “A” en “s”

`s.replaceAll("a{2,3}", "A") → String`

Comprobar si la cadena “s” comienza por la cadena “s1”

`s.startsWith(s1) → boolean`

Obtener la subcadena de “s” a partir de la posición 2 (tercera letra) incluida hasta la posición 4 (quinta letra) sin incluir

`s.substring(2, 4) → String`

Obtener una cadena de “s” quitando los espacios por el principio y por el final

`s.trim() → String`

Obtener una cadena reemplazando la primera subcadena que cumple la expresión regular “a{2,3}” por “A” en “s”

`s.replaceFirst("a{2,3}", "A") → String`

Obtener la subcadena de “s” a partir de la posición 2 (tercera letra) inclusive

`s.substring(2) → String`

Convertir una cadena “s” a minúsculas

`s.toLowerCase() → String`

Convertir una cadena “s” a mayúsculas

`s.toUpperCase() → String`

La clase StringBuffer

En Java se tiene una clase “StringBuffer” definida en el paquete “java.lang”, que permite manejar cadenas

Creación de una cadena buffer

```
s = new StringBuffer("Hola");
```

Convertir una cadena buffer en una cadena

```
s.toString() → String
```

Añadir a la cadena buffer “s” por el final cualquier valor de un tipo “boolean”, “char”, “char[]”, “int”, “long”, “float”, “double”, “Object”, “String”, “StringBuffer”

```
s.append(2) → StringBuffer
```

Borrar de la cadena buffer “s” desde la posición 2 incluida hasta la posición 4 sin incluir

```
s.delete(2,4) → StringBuffer
```


La clase StringBuffer (II)

Insertar en la posición 2 (a partir de la segunda letra) de la cadena buffer “s” cualquier valor de un tipo “boolean”, “char”, “char[]”, “int”, “long”, “float”, “double”, “Object”, “String”, “StringBuffer”

```
s.insert(2, 45.8) → StringBuffer
```

Obtener la cadena inversa de “s”

Cambiar el segundo carácter de “s” por la letra ‘c’

```
s.reverse() → StringBuffer
```

```
s.setCharAt(1, 'c') → void
```

Un ejemplo para invertir una cadena sería

```
// "s" String
```

```
s = new StringBuffer(s).reverse().toString();
```

Las clases String y StringBuffer

Hágase un programa que lea una línea, quite los espacios del principio y del final, ponga las letras en mayúsculas y quite todos los caracteres que no sean letras

```
//Variables
```

```
String s = "";
```

Salida
consola



ESCRIBE UNA LINEA:
HOLACÓMOESTÁS

Hola ¿Cómo estás?

```
s = s.trim().toUpperCase().replaceAll("[^A-ZÑÁÉÍÓÚÛ]", "");
```

```
//Programa
```

```
s = in.leerLine();
```

```
s = s.trim();
```

```
s = s.toUpperCase();
```

```
s = s.replaceAll("[^A-ZÑÁÉÍÓÚÛ]", "");
```

```
System.out.println("*"+s+"*");
```



Se quita los espacios del principio y final de "s" y se devuelve una cadena "s1" que se pone en mayúsculas y se devuelve una cadena "s2" a la que se le quita los caracteres que no son letras.