Centro Asociado Palma de Mallorca

Introducción Práctica de Programación Java

Antonio Rivero Cuesta

Sesión

III

La Sintaxis de Java II

Estructuras de control

- Estructuras de selección
- Estructuras de repetición

Estructuras de selección

- if
- if-else
- Operador condicional
- switch

Sentencia if

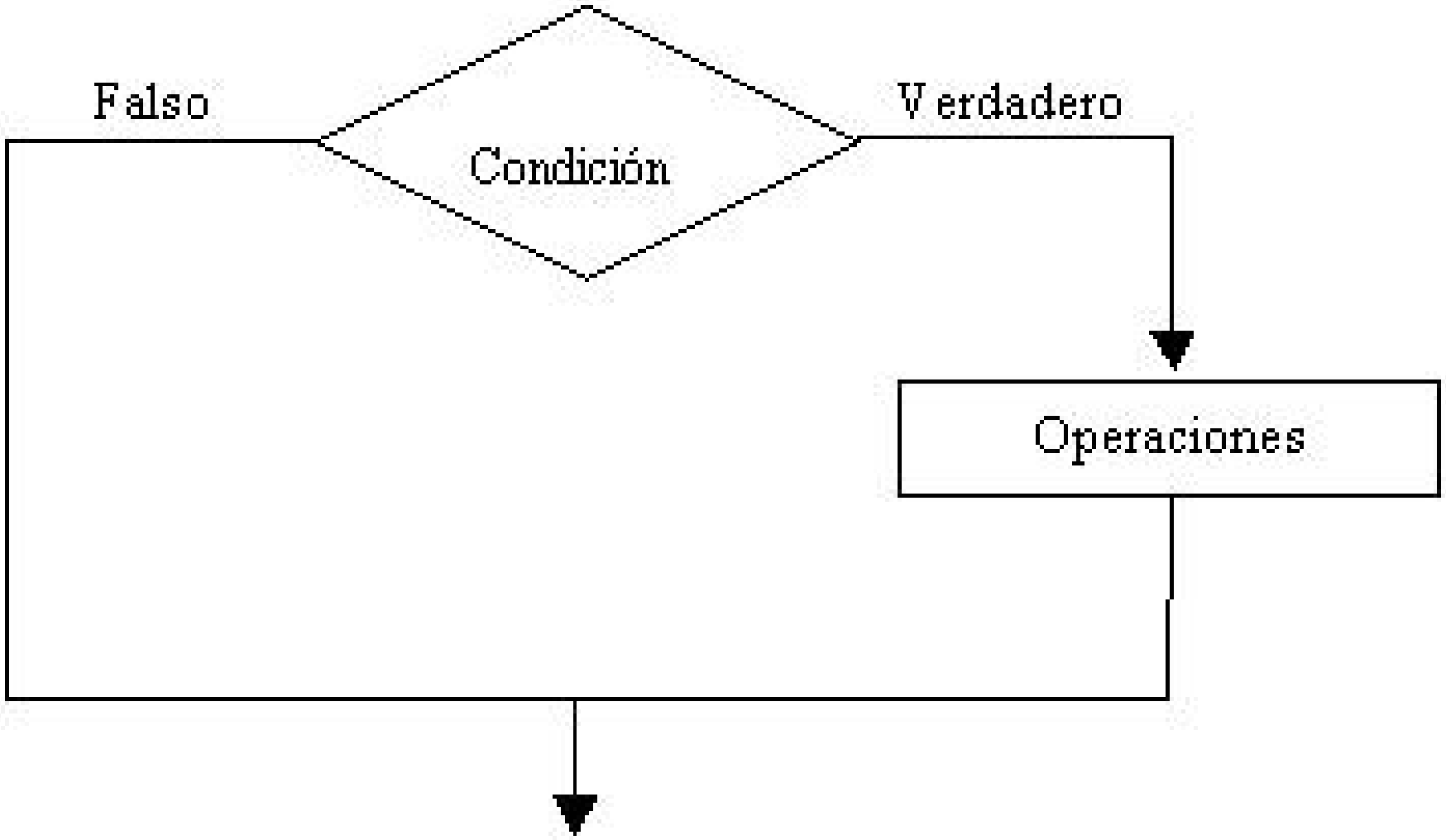
La sentencia **`if`** permite en un programa tomar la decisión sobre la ejecución o no de una acción o de un grupo de acciones, mediante la evaluación de una expresión lógica o booleana.

La acción o grupo de acciones se ejecutan cuando la condición es cierta.

En caso contrario no se ejecutan y se saltan.

Sentencia if

```
if (condición){
    sentencias
}
```


Falso
Condición
Verdadero
Operaciones

Sentencia if - else

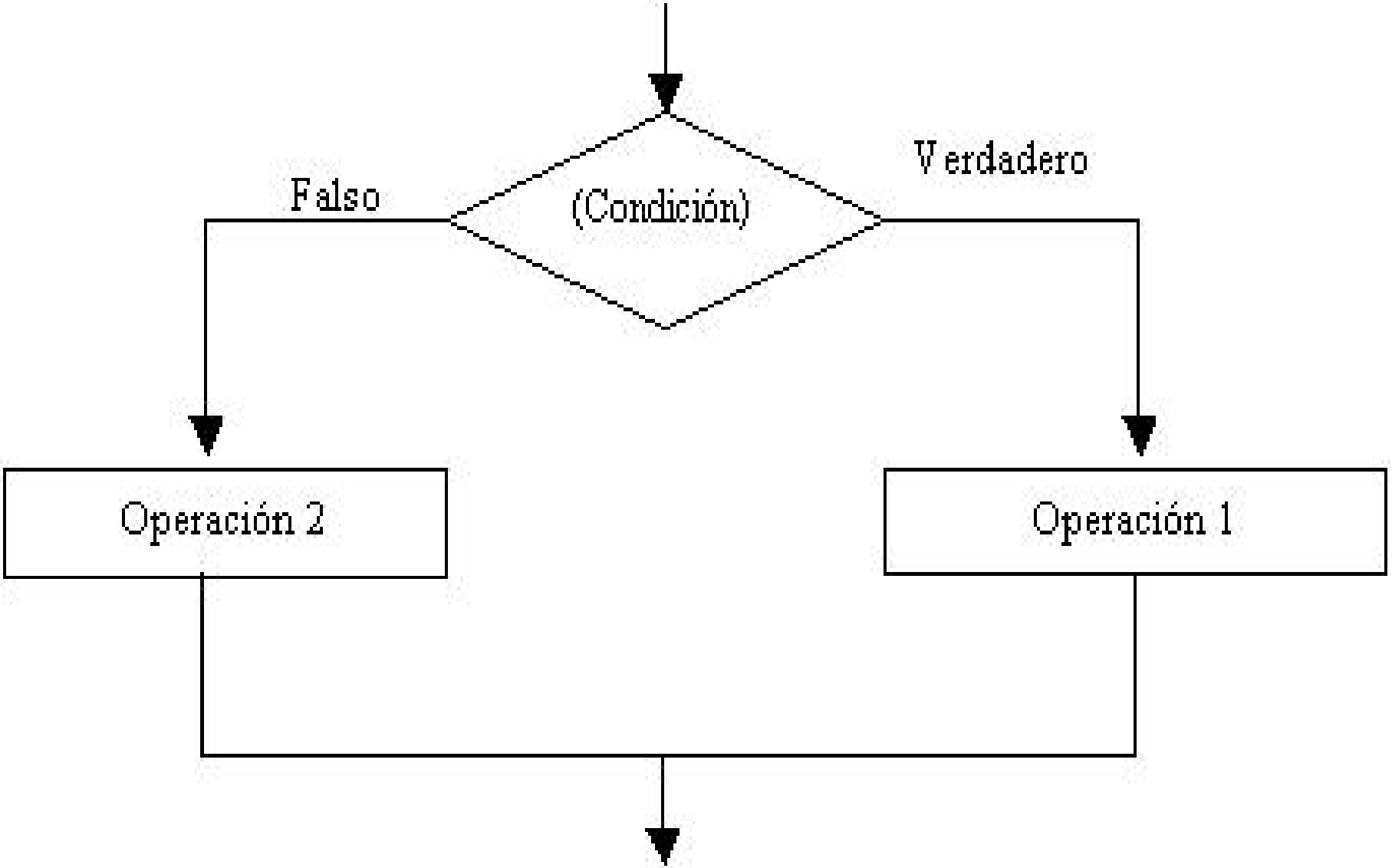
Esta clase de sentencia **`if`** ofrece dos alternativas a seguir, basadas en la comprobación de la condición.

La palabra reservada **`else`** separa las sentencias utilizadas para ejecutar cada alternativa.

Si la evaluación de la condición es verdadera, se ejecuta la sentencia 1 o secuencia de sentencias 1, mientras que si la evaluación es falsa se ejecuta la sentencia 2 o secuencia de sentencias 2.

Sentencia if - else

```
if (condición){
    sentencia 1
}
else {
    sentencia 2
}
```


Falso
(Condición)
Verdadero
Operación 2
Operación 1

Operador condicional

El operador condicional **`?:`** está relacionado con la estructura **`if-else`**.

El operador evalua la condición a la izquierda del símbolo **`?`**.

Si la condición vale **`true`** devuelve el valor de la expresión que haya entre el **`?`** y el **`:`**.

Si la condición vale **`false`** devuelve el valor de la expresión tras el símbolo **`:`**.

Para escribir el mayor de tres números se puede escribir:

```
System.out.print("El número mayor es el de valor");

System.out.println(a > b ? a : b);
```

Sentencia switch

Cuando se tienen muchas alternativas posibles a elegir, el uso de sentencias **`if else-if`** puede resultar bastante complicado, siendo en general más adecuado en estos casos el empleo de la sentencia **`switch`**.

La sintaxis de una sentencia **`switch`** es la siguiente:

Sentencia switch

```
switch (expresión) {
      case valor1:
            sentencias;
            break;
      case valor2:
      case valor3:
            sentencias;
            break;
     ……
      default:
            sentencias;
            break;
}
```

La expresión, que es obligatorio que esté entre paréntesis, tiene que evaluarse a un entero, un carácter, un enumerado o un booleano.

A continuación, en cada **`case`** aparece un valor que únicamente puede ser una expresión constante, es decir, una expresión cuyo valor se puede conocer antes de empezar a ejecutar el programa del mismo tipo que la expresión del **`switch.`**

Después de cada **`case`** se puede poner una única sentencia o un conjunto de ellas.

Los valores asociados en cada **`case`** se comparan en el orden en que están escritos.

Cuando se quiere interrumpir la ejecución de sentencias se utiliza la sentencia **`break`** que hace que el control del programa termine el **`switch`** y continúe ejecutando la sentencia que se encuentre después de esta estructura.

Si no coincide el valor de ningún case con el resultado de la expresión, se ejecuta la parte **`default`**.

Si ningún valor de los **`case`** coincide con el resultado de la expresión y la parte **`default`** no existe, ya que es opcional, no se ejecuta nada de la estructura **`switch`**.

Estructuras de repetición

- while
- do-while
- for

Sentencia while

El bucle **`while`** ejecuta una sentencia o bloque de sentencias mientras se cumple una determinada condición.

La condición tiene que estar obligatoriamente entre paréntesis.

La condición es una expresión lógica.

Si la condición vale **`true`**, se ejecutan las sentencias que componen el bucle.

Cuando concluye la ejecución de las instrucciones del bucle se vuelve a evaluar la condición.

De nuevo, si la condición es cierta se vuelven a ejecutar las instrucciones del bucle.

En algún momento la condición valdrá **`false`**, en cuyo caso finaliza la ejecución del bucle y el programa continúa ejecutándose por la sentencia que se encuentre a continuación de la estructura **`while`**.

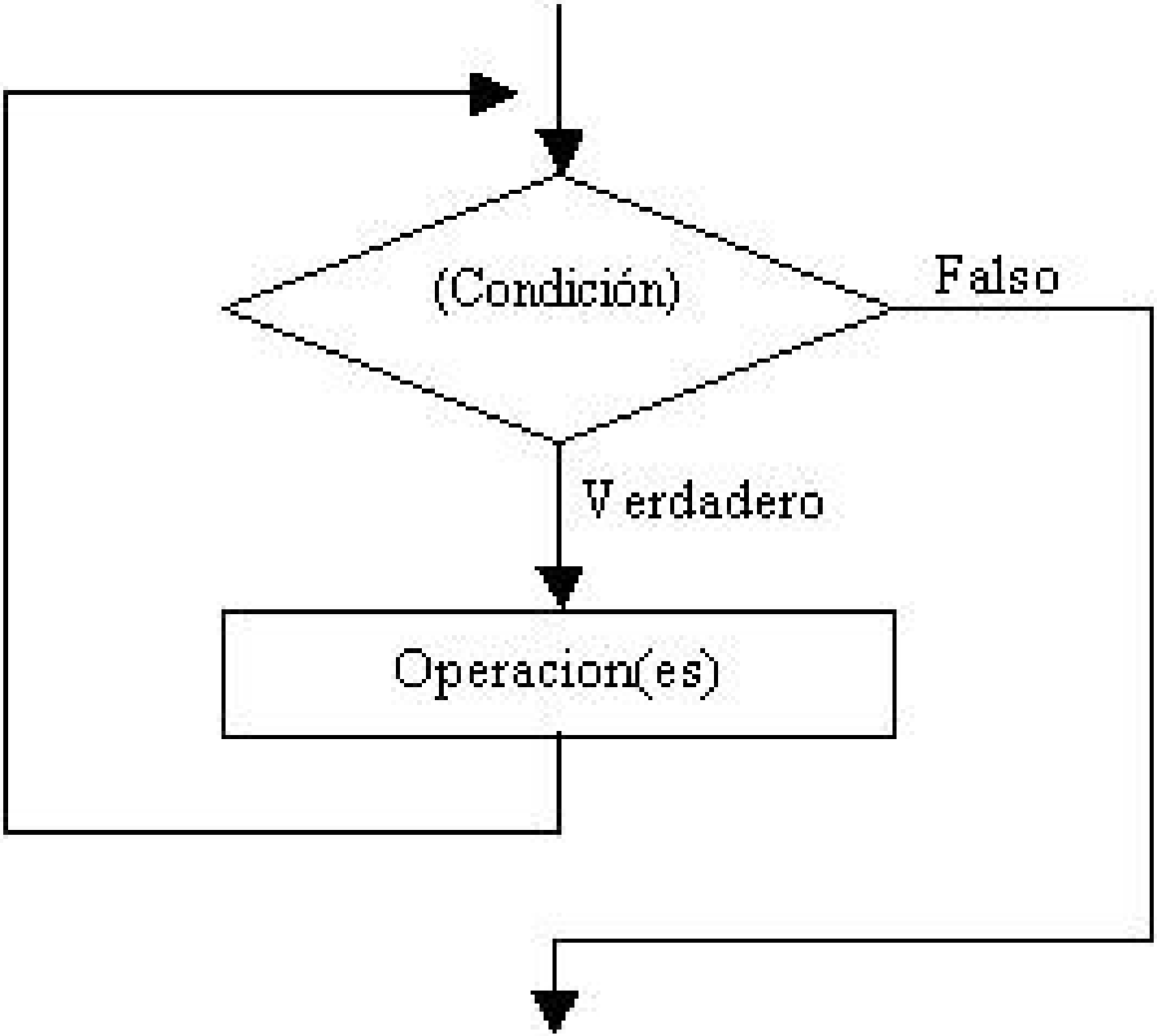
Un problema frecuente en programación se produce cuando aparecen bucles infinitos.

Un bucle infinito es aquel que nunca termina.

Los bucles **`while`** infinitos se producen debido a que la condición que se comprueba nunca se hace falsa, de modo que el bucle **`while`** ejecuta repetidamente sus sentencias una y otra vez.

Sentencia while

```
while (condición){
      sentencias
}
```


(Condición)
Falso
Verdadero
Operacion(es)

Sentencia do-while

La sentencia **`do-while`** es similar a la sentencia **`while`**, excepto que la condición se comprueba después de que el bloque de sentencias se ejecute.

La sentencia o sentencias se ejecutan y, a continuación, se evalúa la condición.

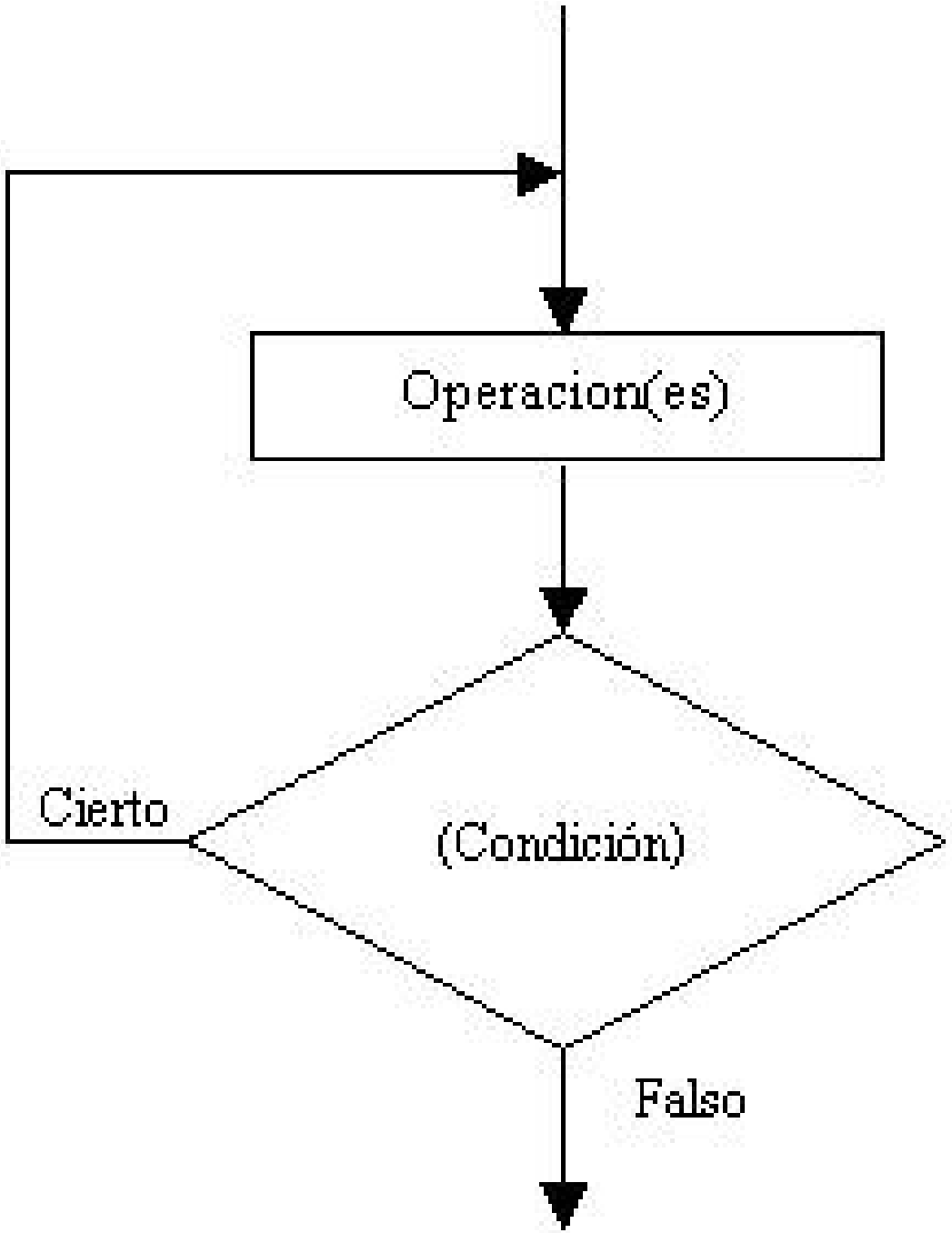
Si la condición se evalúa a un valor verdadero, las sentencias se ejecutan de nuevo.

Este proceso se repite hasta que expresión se evalúa a un valor falso, en cuyo momento se sale de la sentencia **`do-while`**.

Dado que el test condicional se realiza al final del bucle la sentencia o bloque de sentencias se ejecuta al menos una vez.

Sentencia do-while

```
do{
sentencias
}while(condición);
```


Operacion(es)
Cierto
(Condición)
Falso

Sentencia for

El bucle **`for`** está diseñado para ejecutar una secuencia de sentencias un número fijo de veces.

La sintaxis de la sentencia **`for`** es:

Sentencia for

```
for (inicialización ; condición ; incremento){
sentencias
}
```

Las ***sentencias*** podrán ser cero, una única sentencia o un bloque, y serán lo que se repita durante el proceso del bucle.

La inicialización fija los valores iniciales de la variable o variables de control antes de que el bucle **`for`** se procese y ejecute solo una vez.

Si se desea inicializar más de un valor, se puede utilizar un operador especial de los bucles **`for`** en Java, el operador coma, para pegar sentencias.

Cuando no se tiene que inicializar, se omite este apartado; sin embargo, nunca se debe omitir el punto y coma que actúa como separador.

La condición de terminación se comprueba antes de cada iteración del bucle y éste se repite mientras que dicha condición se evalúe a un valor verdadero.

Si se omite no se realiza ninguna prueba y se ejecuta siempre la sentencia **`for`**.

El incremento se ejecuta después de que se ejecuten las sentencias y antes de que se realice la siguiente prueba de la condición de terminación.

Normalmente esta parte se utiliza para incrementar o decrementar el valor de más variables de control y, al igual que en la inicialización, se puede usar en ella el operador coma para pegar sentencias.

Cuando no se tienen valores a incrementar se puede suprimir este apartado.

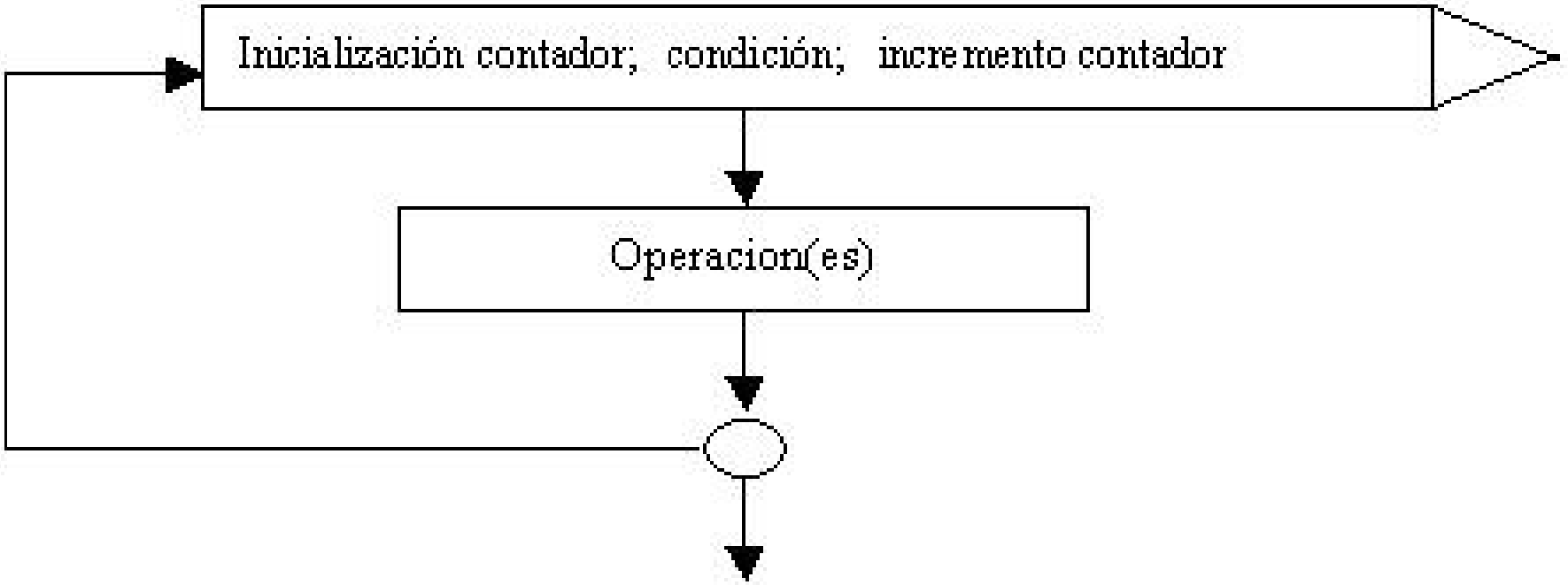
En esencia, el bucle **`for`** comprueba si la condición de terminación es verdadera

Si la condición es Verdadera, se ejecutan las sentencias del interior del bucle, y si la condición es falsa, se saltan todas las sentencias del interior del bucle, es decir, no se ejecutan.

Cuando la condición es verdadera, el bucle ejecuta una iteración, todas sus sentencias, y a continuación la variable de control del bucle se incrementa.

Sentencia for

```
for (declaración-de-variables : colección){
sentencias
}
```


Inicialización contador; condición; incremento contador
Operacion(es)

Uso de las estructuras de repetición

Es importante utilizar el tipo de bucle más apropiado en cada parte de un programa.

Estructura	Usar si:
`While`	Si el bucle se ejecuta 0 o más veces.
`do-while`	Si la parte de ejecución del bucle se ha de hacer al menos una vez.
`for`	Si se sabe el número de veces que se ha de repetir el bucle. Si utilizar la inicialización y la actualización del bucle permite escribir el código de forma más clara. Se realiza un recorrido en una estructura de almacenamiento. Si la estructura de almacenamiento se va a recorrer completa realizando operaciones con sus valores, se utilizará la segunda versión de `for`.

Vectores

Un vector es una estructura de datos que permite almacenar un conjunto de datos del mismo tipo.

Existen dos formas equivalentes de declarar vectores o arrays en Java:

```
int[] nombreDelVector1;
```

```
int nombreDelVector2[];
```

Creación de un vector

Los vectores son objetos, para utilizarlos primero hay que crearlos.

El valor inicial por defecto es **`null`**.

```
nombreDelVector1 = new int[20];
```

```
nombreDelVector2 = new int[100];
```

El vector nombreDelVector1 permite guardar 20 enteros.

El vector nombreDelVector2 permite guardar 100 enteros.

Al crear el vector, cada uno de los elementos se inicializa al valor por defecto:

- 0 para los números.
- **`false`** para los boolean.
- \u0000 para los caracteres
- **`null`** para las referencias a objetos.

Los vectores se pueden crear cuando se declaran:

```
int[] nombreDelVector1 = new int[20];

int nombreDelVector2[] = new int[100];
```

Inicialización estática

Es posible inicializar los elementos del vector a la vez que se crean:

```
String[] díasSemana = {“Lunes”, “Martes”,
“Miércoles”, “Jueves”, “Viernes”, “Sábado”,
“Domingo”};
```

Cuando se desea acceder a los valores de un vector de tamaño N, el primer elemento está en la posición 0 y el último en N−1.

Tamaño del vector

Para conocer el número de elementos de un vector se utiliza su atributo **`length`**.

`System.out.println`(“El vector díasSemana tiene” + **`díasSemana.length`** + “elementos”);

Matrices

Un vector declarado de la siguiente forma representa una tabla de dos dimensiones:

```
int[][] tabla;
```

Para crearlo:

```
int[][] tabla = new int[4][7]
```

Una matriz es una estructura de datos que permite almacenar un conjunto de datos del mismo tipo.

Con un único nombre se define la matriz y por medio de dos subíndices hacemos referencia a cada elemento de la misma.