Estructuras de control repetitivas/iterativas

A menudo es necesario ejecutar
una instrucción o un bloque de instrucciones más de una vez.

Ejemplo

Implementar un programa que calcule
la suma de N números leídos desde teclado.

Podríamos escribir un programa en el que apareciese repetido el código que deseamos que se ejecute varias veces, pero…

- Nuestro programa podría ser demasiado largo.
- Gran parte del código del programa estaría duplicado, lo que dificultaría su mantenimiento en caso de que tuviésemos que hacer cualquier cambio, por trivial que fuese éste.
- Una vez escrito el programa para un número determinado de repeticiones (p.ej. sumar matrices 3x3), el mismo programa no podríamos reutilizarlo si necesitásemos realizar un número distinto de operaciones (p.ej. sumar matrices 4x4).

Las **estructuras de control repetitivas o iterativas**, también conocidas como "**bucles**", nos permiten resolver de forma elegante este tipo de problemas. Algunas podemos usarlas cuando conocemos el número de veces que deben repetirse las operaciones. Otras nos permiten repetir un conjunto de operaciones mientras se cumpla una condición.

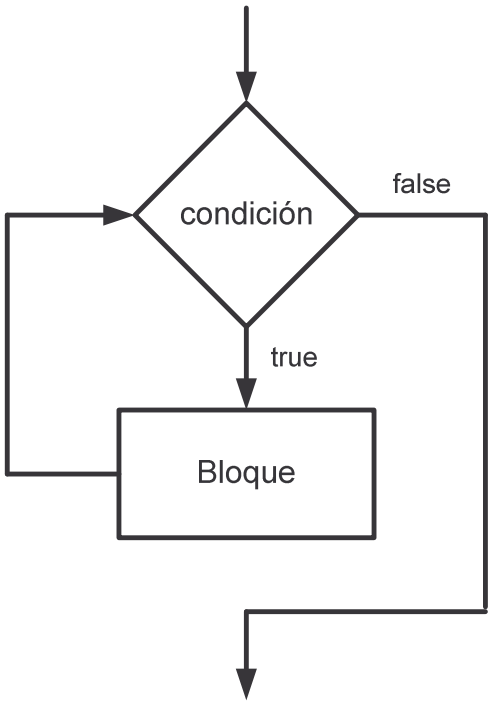
Iteración: Cada repetición de las instrucciones de un bucle.

El bucle **`while`**

Permite repetir la ejecución de un conjunto de sentencias mientras se cumpla una condición:

```
while (condición)
      sentencia;
```

```
while (condición) {
      bloque
}
```



El bucle `while` terminará su ejecución
cuando deje de verificarse la condición que controla su ejecución.

Si, inicialmente, no se cumple la condición,
el cuerpo del bucle no llegará a ejecutarse.

MUY IMPORTANTE

En el cuerpo del bucle debe existir algo que haga variar el valor asociado a la condición que gobierna la ejecución del bucle.

Ejemplo
Tabla de multiplicar de un número

```
public class While1
{
  public static void main( String args[] )
  {
    int n; // Número
    int i; // Contador

    n = Integer.parseInt( args[0] );
    i = 0;

    while (i<=10) {
      System.out.println (n+" x "+i+" = "+(n*i));
      i++;
    }
  }
}
```

Ejemplo
Divisores de un número

```
public class While2
{
  public static void main( String args[] )
  {
    int n;
    int divisor;

    n = Integer.parseInt( args[0] );

    System.out.println("Los divisores son:");

    divisor = n;

    while (divisor>0) {

      if ((n%divisor) == 0)
         System.out.println(divisor);

      divisor--;
    }
  }
}
```

En los ejemplos anteriores,
se conoce de antemano el número de iteraciones
que han de realizarse (cuántas veces se debe ejecutar el bucle):

La expresión del `while` se convierte en una simple comprobación del valor de una variable contador.

El contador es una variable que se incrementa o decrementa en cada iteración y nos permite saber la iteración en la que nos encontramos en cada momento.

En el cuerpo del bucle, siempre se incluye una sentencia

```
contador++;
```

o bien

```
contador--;
```

para que, eventualmente,
la condición del `while` deje de cumplirse.

En otras ocasiones, puede que no conozcamos de antemano cuántas iteraciones se han de realizar.

La condición del `while` puede que tenga un aspecto diferente pero, en el cuerpo del bucle, deberá seguir existiendo algo que modifique el resultado de evaluar la condición.

Ejemplo
Sumar una serie de números
hasta que el usuario introduzca un cero

```
import javax.swing.JOptionPane;

public class While3
{
  public static void main( String args[] )
  {
    float  valor;
    float  suma;

    suma = 0;
    valor = leerValor();

    while (valor!=0) {
          suma += valor;
          valor = leerValor();
    }

    mostrarValor("Suma de los datos", suma);
    System.exit(0);
  }

  private static float leerValor ()
  {
    String entrada;

    entrada = JOptionPane.showInputDialog
              ( "Introduzca un dato:" );

    return Float.parseFloat(entrada);
  }

  private static void mostrarValor
                      (String mensaje, float valor)
  {
    JOptionPane.showMessageDialog
      ( null, valor, mensaje,
        JOptionPane.INFORMATION_MESSAGE );
  }
}
```

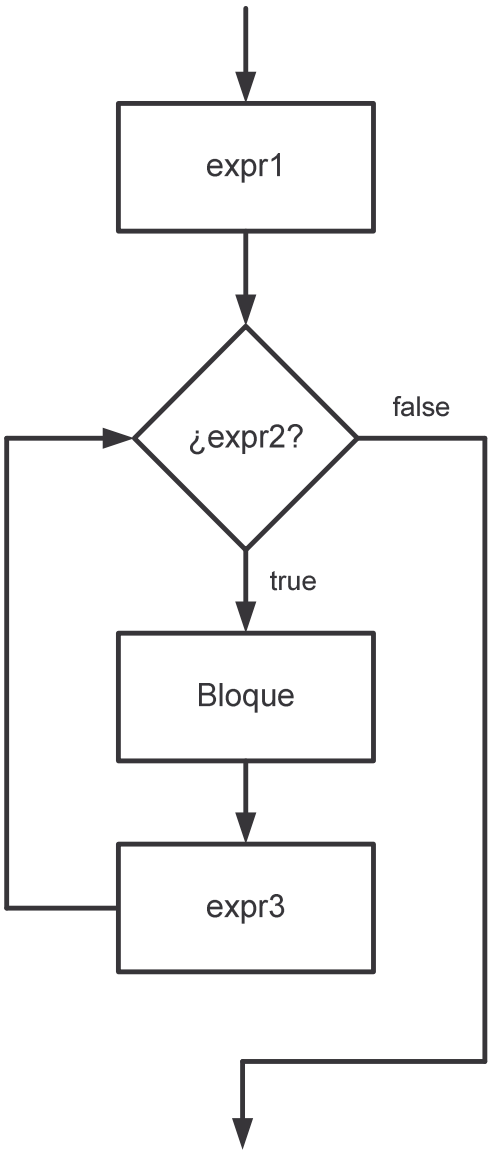
El valor introducido determina en cada iteración
si se termina o no la ejecución del bucle.

El bucle **`for`**

Se suele emplear en sustitución del bucle `while`
cuando se conoce el número de iteraciones que hay que realizar.

Sintaxis

```
for (expr1; expr2; expr3) {
    bloque;
}
```



Equivalencia entre `for` y `while`

Un fragmento de código
como el que aparecía antes con un bucle `while`:

```
i = 0;
while (i<=10) {
  System.out.println (n+" x "+i+" = "+(n*i));
  i++;
}
```

puede abreviarse si utilizamos un bucle `for`:

```
for (i=0; i<=10; i++) {
  System.out.println (n+" x "+i+" = "+(n*i));
}
```

Como este tipo de estructuras de control es muy común,
el lenguaje nos ofrece una forma más compacta
de representar un bucle cuando sabemos
cuántas veces ha de ejecutarse el cuerpo del bucle.

En general,

```
for (expr1; expr2; expr3) {
    bloque;
}
```

equivale a

```
expr1;
while (expr2) {
      bloque;
      expr3;
}
```

Ejemplo
Cálculo del factorial de un número

Bucle **for**

```
public class FactorialFor
{
  public static void main( String args[] )
  {
    long i,n,factorial;

    n = Integer.parseInt( args[0] );

    factorial = 1;

    for (i=1; i<=n; i++) {
        factorial *= i;
    }

    System.out.println ( "f("+n+") = " + factorial);
  }
}
```

Bucle **while**

```
public class FactorialWhile
{
  public static void main( String args[] )
  {
    long i,n,factorial;

    n = Integer.parseInt( args[0] );

    factorial = 1;
    i = 1;

    while (i<=n) {
        factorial *= i;
        i++;
    }

    System.out.println ( "f("+n+") = " + factorial);
  }
}
```

```
for (expr1; expr2; expr3) {
    bloque;
}
```

En un bucle `for`

La primera expresión, `expr1`, suele contener inicializaciones de variables separadas por comas. En particular, siempre aparecerá la inicialización de la variable que hace de contador.

- Las instrucciones que se encuentran en esta parte del `for` sólo se ejecutarán una vez antes de la primera ejecución del cuerpo del bucle (`bloque`).

La segunda expresión, `expr2`, es la que contiene una expresión booleana (la que aparecería en la condición del bucle `while` equivalente para controlar la ejecución del cuerpo del bucle).

La tercera expresión, `expr3`, contiene las instrucciones, separadas por comas, que se deben ejecutar al finalizar cada iteración del bucle (p.ej. el incremento/decremento de la variable contador).

El bloque de instrucciones `bloque` es el ámbito del bucle (el bloque de instrucciones que se ejecuta en cada iteración).

Cabecera del bucle	Número de iteraciones
`for (i=0; i<N; i++)`	N
`for (i=0; i<=N; i++)`	N+1
`for (i=k; i<N; i++)`	N-k
`for (i=N; i>0; i--)`	N
`for (i=N; i>=0; i--)`	N+1
`for (i=N; i>k; i--)`	N-k
`for (i=0; i<N; i+=k)`	N/k
`for (i=j; i<N; i+=k)`	(N-j)/k
`for (i=1; i<N; i*=2)`	$\log_2 N$
`for (i=0; i<N; i*=2)`	∞
`for (i=N; i>0; i/=2)`	$\log_2 N + 1$

suponiendo que N y k sean enteros positivos

Bucles infinitos

Un bucle infinito es un bucle que se repite "infinitas" veces:

```
for (;;)          /*bucle infinito*/

while (true)      /*bucle infinito*/
```

Si nunca deja de cumplirse la condición del bucle, nuestro programa se quedará indefinidamente ejecutando el cuerpo del bucle, sin llegar a salir de él.

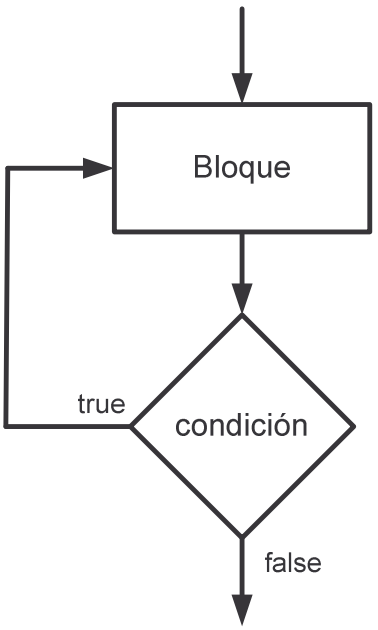
El bucle `do while`

Tipo de bucle, similar al `while`, que realiza la comprobación de la condición después de ejecutar el cuerpo del bucle.

Sintaxis

```
do
  sentencia;
while (condición);
```

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



- El bloque de instrucciones se ejecuta, al menos, una vez.

- El bucle `do while` resulta especialmente indicado para validar datos de entrada (comprobar que los valores de entrada obtenidos están dentro del rango de valores que el programa espera).

En todos nuestros programas debemos asegurarnos de que se obtienen datos de entrada válidos antes de realizar cualquier tipo de operación con ellos.

Ejemplo
Cálculo del factorial
comprobando el valor del dato de entrada

```
import javax.swing.JOptionPane;

public class FactorialDoWhile
{
  public static void main( String args[] )
  {
    long n;

    do {
       n = leerEntero(0,20);
    } while ((n<0) || (n>20));

    mostrarMensaje ( "f("+n+") = "+ factorial(n));
    System.exit(0);
  }

  private static long factorial (long n)
  {
    long i;
    long factorial = 1;

    for (i=1; i<=n; i++) {
        factorial *= i;
    }

    return factorial;
  }

  private static int leerEntero (int min, int max)
  {
    String entrada = JOptionPane.showInputDialog
                     ( "Introduzca un valor entero"
                     + " (entre "+min+" y "+max+"):" );

    return Integer.parseInt(entrada);
  }

  private static void mostrarMensaje (String mensaje)
  {
    JOptionPane.showMessageDialog(null,mensaje);
  }
}
```

Ejemplo
Cálculo de la raíz cuadrada de un número

```
import javax.swing.JOptionPane;

public class Sqrt
{
  public static void main( String args[] )
  {
    double n;
    do {
       n = leerReal ("Introduzca un número positivo");
    } while (n<0);
    mostrarMensaje( "La raíz cuadrada de "+n
                  + " es aproximadamente "+raiz(n));
    System.exit(0);
  }

  private static double raiz (double n)
  {
    double r;     // Raíz cuadrada del número
    double prev;  // Aproximación previa de la raíz

    r = n/2;
    do {
       prev = r;
       r = (r+n/r)/2;
    } while (Math.abs(r-prev) > 1e-6);
    return r;
  }

  private static double leerReal (String mensaje)
  {
    String entrada;
    entrada = JOptionPane.showInputDialog(mensaje);
    return Double.parseDouble(entrada);
  }
  private static void mostrarMensaje (String mensaje)
  {
    JOptionPane.showMessageDialog(null,mensaje);
  }
}
```

Bucles anidados

Los bucles también se pueden anidar:

```
for (i=0; i<N;i++) {
    for (j=0; j<N; j++) {
        printf("(%d,%d) ",i,j);
    }
}
```

genera como resultado:

```
(0,0) (0,1) (0,2) … (0,N)
(1,0) (1,1) (1,2) … (1,N)
                …
(N,0) (N,1) (N,2) … (N,N)
```

Ejemplo

```
int n,i,k;

…
n = 0;                          // Paso 1
for (i=1; i<=2; i++) {          // Paso 2
    for (k=5; k>=1; k-=2) {     // Paso 3
        n = n + i + k;          // Paso 4
    }                           // Paso 5
}                               // Paso 6
…                               // Paso 7
```

Paso	1	2	3	4	5	3	4	5	3	4	5	3	6	2	3	4	5	3	4	5	3	4	5	3	6	2	7
N	0			6			10			12						19			24			27					
i	?	1											2												3		
k	?		5		3			1			–1				5		3			1			–1				