UD02.2.Estructuras de Control. Bucles Apuntes

DAM1-Programación 2023-24

Bucles controlados por condición: while y do...while

Introducción

Un bucle es un tipo de estructura que contiene un bloque de instrucciones que se ejecuta repetidas veces; cada ejecución o repetición del bucle se llama *iteración*.

El uso de bucles simplifica la escritura de programas, minimizando el código duplicado. Cualquier fragmento de código que sea necesario ejecutar varias veces seguidas es susceptible de incluirse en un bucle. Java dispone de los bucles: `while`, `do-while` y `for`.

3.1. Bucles controlados por condición

El control del número de iteraciones se lleva a cabo mediante una condición. Si la evaluación de la condición es cierta, el bucle realizará una nueva iteración.

3.1.1. while

Al igual que la instrucción `if`, el comportamiento de `while` depende de la evaluación de una condición. El bucle `while` decide si realizar una nueva iteración basándose en el valor de la condición. Su sintaxis es:

```
while (condición) {
   bloque de instrucciones
   ...
}
```

El comportamiento de este bucle (véase Figura 3.1) es:

1. Se evalúa `condición`.
2. Si la evaluación resulta `true`, se ejecuta el bloque de instrucciones.
3. Tras ejecutarse el bloque de instrucciones, se vuelve al primer punto.
4. Si, por el contrario, la condición es `false`, terminamos la ejecución del bucle.

Por ejemplo, podemos mostrar los números del 1 al 3 mediante un bucle `while` controlado por la variable `i`, que empieza valiendo 1, con la condición `i <= 3`:

```
int i = 1; //valor inicial
while (i <= 3) { //el bucle iterará mientras i sea menor o igual que 3
   System.out.println(i); //mostramos i
   i++; //incrementamos i
}
```

Veamos una traza de la ejecución:

1. Se declara la variable `i` y se le asigna el valor 1.
2. La instrucción `while` evalúa la condición (`i <= 3`): ¿es $1 \leq 3$? Cierto.
3. Se ejecuta el bucle de instrucciones: `System.out.println` e `i++`. Ahora la `i` vale 2.

4. La instrucción `while` vuelve a evaluar la condición: ¿es 2 ≤ 3? Cierto.
5. Se ejecuta el bloque de instrucciones: `System.out.println` e `i++`. Ahora la `i` vale 3.
6. La instrucción `while` vuelve a evaluar la condición: ¿es 3 ≤ 3? Cierto.
7. Se ejecuta el bloque de instrucciones: `System.out.println` e `i++`. Ahora la `i` vale 4.
8. La instrucción `while` vuelve a evaluar la condición: ¿es 4 ≤ 3? Falso.
9. Se termina el bucle y se pasa a ejecutar la instrucción siguiente.

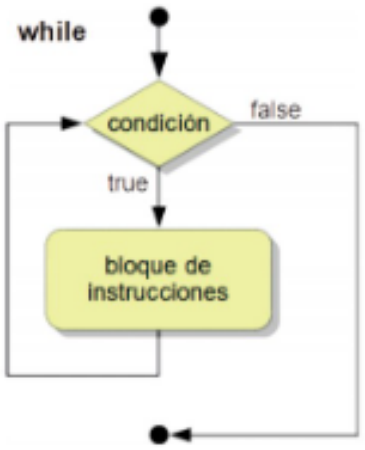

Figura 3.1. *Bucle* `while`.

Un bucle `while` puede realizar cualquier número de iteraciones, desde cero, cuando la primera evaluación de la condición resulta falsa, hasta infinitas, en el caso de que la condición sea siempre cierta. Esto es lo que se conoce como *bucle infinito*.

Veamos un ejemplo de un bucle `while` que nunca llega a ejecutarse:

```
int cuentaAtras = -8; //valor negativo
   while (cuentaAtras >= 0) {
   ...
}
```

En este caso, independientemente del bloque de instrucciones asociado a la estructura `while`, no se llega a entrar nunca en el bucle, debido a que la condición no se cumple ni siquiera la primera vez. Se realizan cero iteraciones. En cambio,

```
int cuentaAtras = 10;
while (cuentaAtras >= 0) {
   System.out.println(cuentaAtras);
}
```

Dentro del bloque de instrucciones no hay nada que modifique la variable `cuentaAtras`, lo que hace que la condición permanezca idéntica, evaluándose siempre `true` y haciendo que el bucle sea infinito.

Actividad propuesta 3.1

Diseña una aplicación que muestre la edad máxima y mínima de un grupo de alumnos. El usuario introducirá las edades y terminará escribiendo un –1.

E0301. Diseñar un programa que muestre, para cada número introducido por teclado, si es par, si es positivo y su cuadrado. El proceso se repetirá hasta que el número introducido sea 0.

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    int numero;

    // Entrada de datos
    Scanner sc = new Scanner(System.in);

    // Entrada de datos
    System.out.println("Introduce un número (0 para salir): ");
    numero = sc.nextInt();

    // Proceso
    while (numero != 0) {
        // Operaciones
        if (numero % 2 == 0) {
            System.out.print("Es par. ");
        } else {
            System.out.print("Es impar. ");

        }

        if (numero < 0) {
            System.out.print("Es negativo. ");
        } else if (numero > 0) {
            System.out.print("Es positivo. ");
        } else {
            System.out.println("Es cero. ");
        }

        System.out.println("Su cuadrado es " +  (int) Math.pow(numero, 2));

        System.out.println("Introduce un número (0 para salir): ");
        numero = sc.nextInt();
    } // while
}
```

E0302. Implementar una aplicación para calcular datos estadísticos de las edades de los alumnos de un centro educativo. Se introducirán datos hasta que uno de ellos sea negativo, y se mostrará: la suma de todas las edades introducidas, la media, el número de alumnos y cuántos son mayores de edad.

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    int edad;
    int sumEdades = 0, mediaEdades, numAlumnos = 0, numMayores18 = 0;

    // Entrada de datos
    Scanner sc = new Scanner(System.in);
    System.out.println("EDADES: Sumas, medias, etc.");
    System.out.println("Introduce las edades de los alumnos (número negativo para
terminar)");
```

```
    edad = sc.nextInt();

    while (edad >= 0) {
        numAlumnos++; // Contador de alumnos
        sumEdades += edad; // Acumulador de edades
        if (edad >= 18) {
            numMayores18++; // Contador de alumnos > 18
        }
        edad = sc.nextInt();
    }
    mediaEdades = sumEdades / numAlumnos; // Media (entera) de edades

    // Salida
    System.out.println("Número de alumnos: " + numAlumnos);
    System.out.println("Número de alumnos mayores de edad: " + numMayores18);
    System.out.println("Suma de las edades de los alumnos: " + sumEdades);
    System.out.println("Media de las edades de los alumnos: " + mediaEdades);
}
```

E0303. Codificar el juego “el número secreto”, que consiste en acertar un número entre 1 y 100 (generado aleatoriamente). Para ello se introduce por teclado una serie de números, para los que se indica: “mayor” o “menor”, según sea mayor o menor con respecto al número secreto. El proceso termina cuando el usuario acierta o cuando se rinde (introduciendo un -1).

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    final int NUM_MAXIMO = 100;
    int numSecreto = (int)(Math.random()*NUM_MAXIMO + 1);
    int numUsuario;

    // Entrada de datos
    Scanner sc = new Scanner(System.in);
    System.out.println("ADIVINA EL NÚMERO SECRETO (1-" + NUM_MAXIMO + ")");

    do {
        // Cuerpo del bucle
        // Leer número
        System.out.println("Introduce un número: ");
        numUsuario = sc.nextInt();

        // Comparar con número secreto
        if (numUsuario == -1) {
            System.out.println("Te has rendido...");
        } else {
            if (numUsuario > numSecreto) {
               System.out.println("Demasiado alto!");
            } else if (numUsuario < numSecreto) {
               System.out.println("Demasiado bajo!");
            } else {
               System.out.println("Acertaste Enhorabuena!!");
            }
        }
    } while (numUsuario != numSecreto && numUsuario != -1);
}
```

E0304. Un centro de investigación de la flora urbana necesita una aplicación que muestre cuál es el árbol más alto. Para ello introducirá por teclado la altura (en centímetros) de cada árbol (terminando la introducción de datos cuando se utilice el -1 como altura). Los árboles se identifican mediante etiquetas con números únicos correlativos, comenzando en 0. Diseñar una aplicación que resuelva el problema planteado.

3.1.2. do-while

Disponemos de un segundo bucle controlado por una condición: el bucle `do-while`, muy similar a `while`, con la diferencia de que primero se ejecuta el bloque de instrucciones y después se evalúa la condición para decidir si se realiza una nueva iteración. Su sintaxis es:

```
do {
   bloque de instrucciones
   ...
} while (condición);
```

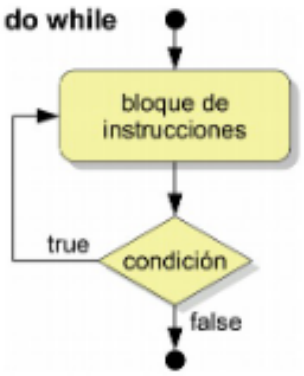

Figura 3.2. *Representación del flujo de ejecución de un bucle* `do-while`.

La Figura 3.2 describe su comportamiento. Se compone de los siguientes puntos:

1. Se ejecuta el bloque de instrucciones.
2. Se evalúa `condición`.
3. Según el valor obtenido, se termina el bucle o se vuelve al punto 1.

Como ejemplo, vamos a escribir el código que muestra los números del 1 al 10 utilizando un bucle `do-while`, en vez de un bucle `while`:

```
int i = 1;
do {
   System.out.println(i);
   i++;
} while (i <= 10);
```

El bucle do-while es el único cuya sintaxis en Java termina en punto y coma (;). Como la condición se evalúa al final, el cuerpo del bucle se ejecutará siempre al menos una vez.

E0305. Desarrollar un juego que ayude a mejorar el cálculo mental de la suma. El jugador tendrá que introducir la solución de la suma de dos números aleatorios comprendidos entre 1 y 100. Mientras la solución sea correcta, el juego continuará. En caso contrario, el programa terminará y mostrará el número de operaciones realizadas correctamente.

Amplía el programa para que muestre el número de aciertos al terminar.

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    final int NUM_MAXIMO = 100;
    int operando1;
    int operando2;
    int numUsuario;
    int numAciertos = 0;

    // Entrada de datos
    Scanner sc = new Scanner(System.in);
    System.out.println("CALCULO MENTAL: SUMAS");

    // Proceso
    do {
        // Obtener dos números aleatorios
        operando1 = (int)(Math.random()*NUM_MAXIMO + 1);
        operando2 = (int)(Math.random()*NUM_MAXIMO + 1);

        // Mostrar al usuario y pedir el resultado
        System.out.println(operando1 + " + " + operando2 + " = ?");
        numUsuario = sc.nextInt();

        // Comparar suma con el resultado del usuario
        if (numUsuario == operando1 + operando2) {
            numAciertos++;
        } else {
            System.out.println("Error! El resultado era " + (int)(operando1 +
operando2));
        }
    } while (numUsuario == operando1 + operando2);

    // Salida
    System.out.println("Has conseguido " + numAciertos + " aciertos.");
}
```

La principal diferencia entre el bucle `while` y el `do-while` es que éste último se ejecuta, al menos, una vez. Además, en cuanto a la sintaxis, es el único que termina en punto y coma (`;`).

Bucles controlados por contador: for

3.2. Bucles controlados por contador: for

El bucle `for` permite controlar el número de iteraciones mediante una variable (que suele recibir el nombre de *contador*). La sintaxis de la estructura `for` es:

```
for (inicialización; condición; incremento) {
   bloque de instrucciones
   ...
}
```

Donde,

- **Inicialización:** es una lista de instrucciones, separadas por comas, donde generalmente se inicializan las variables que van a controlar el bucle. Se ejecutan una sola vez antes de la primera iteración.

- **Condición:** es una expresión booleana que controla las iteraciones del bucle. Se evalúa antes de cada iteración; el bloque de instrucciones se ejecutará solo cuando el resultado sea `true`.
- **Incremento:** es una lista de instrucciones, separadas por comas, donde se suelen modificar las variables que controlan la condición. Se ejecuta al final de cada iteración.

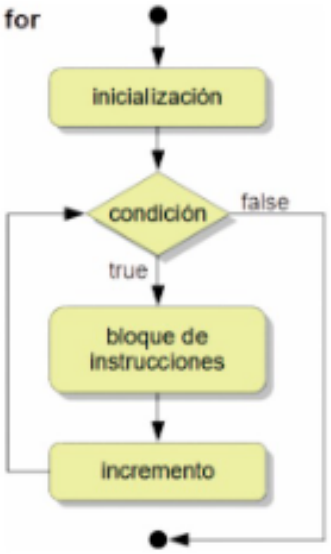

Figura 3.3. *Secuencia del flujo de control de un bucle* `for`.

El funcionamiento de `for` se describe en la Figura 3.3. Consiste en los siguientes puntos:

1. Se ejecuta la inicialización; esto se hace una sola vez al principio.
2. Se evalúa la condición: si resulta `false`, salimos del bucle y continuamos con el resto del programa; en caso de que la evaluación sea `true`, se ejecuta todo el bloque de instrucciones.
3. Cuando termina de ejecutarse el bloque de instrucciones, se ejecuta el incremento.
4. Se vuelve de nuevo al punto 2.

Aunque `for` está controlado por una condición que, en principio, puede ser cualquier expresión booleana, la posibilidad de configurar la inicialización y el incremento de las variables que controlan el bucle permite determinar de antemano el número de iteraciones.

Veamos un ejemplo donde solo se usa la variable `i` para controlar el bucle:

```
for (int i = 1; i <= 2; i++) {
   System.out.println("La i vale " + i);
}
```

Argot técnico

En este caso, la variable `i`, además de inicializarse, también se declara en la zona de inicialización. Esto significa que `i` solo puede usarse dentro de la estructura `for`. En la Unidad 4 profundizaremos en el ámbito de las variables.

A continuación, se muestra una traza de la ejecución del bucle anterior:

1. Primero se ejecuta la inicialización: `i=1`;
2. Evaluamos la condición: ¿es cierto que `i` ≤ 2? Es decir: ¿1 ≤ 2?
3. Cierto. Ejecutamos el bloque de instrucción: `System.out.println(...)`
4. Obtenemos el mensaje: «La i vale 1».
5. Terminado el bloque de instrucciones, ejecutamos el incremento: (`i++`) `i` vale 2.
6. Evaluamos la condición: ¿es cierto que `i` ≤ 2? Es decir: ¿2 ≤ 2?
7. Cierto. Ejecutamos `System.out.println(...)`
8. Obtenemos el mensaje:« La `i` vale 2».
9. Ejecutamos el incremento: (`i++`) la `i` vale 3.
10. Evaluamos la condición: ¿es cierto que `i` ≤ 2? Es decir: ¿3 ≤ 2?
11. Falso. El bucle termina y continúa la ejecución de las sentencias que siguen a la estructura `for`.

E0306. Escribir una aplicación para aprender a contar, que pedirá un número n y mostrará todos los números del 1 al n.

E0307. Escribir todos los múltiplos de 7 menores que 100.

E0308. Pedir diez números enteros por teclado y mostrar la media.
Amplía codificando el número de números como una constante.

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    final int NUM_NUMEROS = 15;
    int numero;
    int suma = 0;
    double media;

    // Entrada de datos
    System.out.println("Vamos a calcular la media de " + NUM_NUMEROS + "
números.");
    for (int i = 1; i <= NUM_NUMEROS; i++) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Escribe un número: ");
        numero = sc.nextInt();
        // suma = suma + numero;
        suma += numero;
    }
    // Proceso
    media = suma / (double) NUM_NUMEROS;

    // Salida
    System.out.printf("La media es %.2f", media);
}
```

E0309. Implementar una aplicación que pida al usuario un número comprendido entre 1 y 10. Hay que mostrar la tabla de multiplicar de dicho número, asegurándose de que el número introducido se encuentra en el rango establecido.

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    int numero;

    // Entrada de datos
    Scanner sc = new Scanner(System.in);
    System.out.println("Escribe un número entre 1 y 10: ");
    numero = sc.nextInt();
    // Si el número no es correcto repetimos la entrada de datos
    // while (numero < 1 || numero > 10) {
    while (!(numero >= 1 && numero <= 10)) {
        System.out.println("El número no está entre 1 y 10.");
        System.out.println("Por favor escribe otro número: ");
        numero = sc.nextInt();
    }

    // Proceso & Salida
    for (int i = 1; i <= 10; i++) {
        System.out.println(numero + " * " + i + " = " + numero * i);
    }
```

E0310. Diseñar un programa que muestre la suma de los 10 primeros números impares.

E0311. Pedir un número y calcular su factorial. Por ejemplo el factorial de 5 se denota 5! y es igual a 5 x 4 x 3 x 2 x 1 = 120.

E0312. Pedir 5 calificaciones de alumnos y decir al final si hay algún suspenso.

E0313. Dadas 6 notas, escribir la cantidad de alumnos aprobados, condicionados (nota igual a 4) y suspensos.

Eco. Implementa una aplicación que pida al usuario un número e imprima por pantalla el texto “Eco…” tantas veces como indique el número introducido.

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    int numero;
    int i; // contador

    // Entrada de datos
    Scanner sc = new Scanner(System.in);
    System.out.println("Escribe un número: ");
    numero = sc.nextInt();

    // Salida
    System.out.println("FOR");
    for (i = 1; i <= numero; i++) {
        // Cuerpo del bucle
        System.out.println(i + " Eco...");
```

```
    }

    System.out.println("WHILE");
    // While
    i = 1;
    while (i <= numero) {
        System.out.println(i + " Eco...");
        i++;
    }

    System.out.println("DO_WHILE");
    i = 1;
    do {
        System.out.println(i + " Eco...");
        i++;
    } while (i <= numero);
}
```

Dígitos. Implementa un programa que pida un número entero y lo muestre dígito a dígito. Por ejemplo, para el número de entrada 123, deberá mostrar por separado los dígitos 3, 2 y 1.

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    long numero;
    long divisor;

    // Entrada de datos
    Scanner sc = new Scanner(System.in);
    System.out.println("Introduce un número entre 1 y 9.223.372.036.854.775.808:
");
    numero = sc.nextInt();

    // Proceso & Salida
    for (int i = 18; i >= 0; i--) {
        divisor = (long) Math.pow(10, i);
        if (numero / divisor != 0) {
            System.out.println((numero / divisor) % 10);
        }
    }
}
```

NúmerosEnTexto. Adapta el ejercicio propuesto 2.14. para imprimir por pantalla los números 1 al 99 en formato texto.

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    int numero;
    int unidades, decenas;
    String textoUnidades = " ";
    String textoDecenas = " ";
    String textoNumero = " ";

    for(numero = 1; numero <= 99; numero++) {
        // Proceso
        unidades = numero % 10;
        decenas = numero / 10;

        // Pasamos a texto las unidades
        switch (unidades) {
            case 1: textoUnidades = "uno"; break;
            case 2: textoUnidades = "dos"; break;
            case 3: textoUnidades = "tres"; break;
            case 4: textoUnidades = "cuatro"; break;
            case 5: textoUnidades = "cinco"; break;
```

```
            case 6: textoUnidades = "seis"; break;
            case 7: textoUnidades = "siete"; break;
            case 8: textoUnidades = "ocho"; break;
            case 9: textoUnidades = "nueve"; break;
        }

        // Pasamos a texto las decenas
        switch (decenas) {
            case 1: textoDecenas = "diez"; break;
            case 2: textoDecenas = "veinte"; break;
            case 3: textoDecenas = "treinta"; break;
            case 4: textoDecenas = "cuarenta"; break;
            case 5: textoDecenas = "cincuenta"; break;
            case 6: textoDecenas = "sesenta"; break;
            case 7: textoDecenas = "setenta"; break;
            case 8: textoDecenas = "ochenta"; break;
            case 9: textoDecenas = "noventa"; break;
        }

        // Evaluamos las decenas para generalizar los casos posibles
        if (decenas == 0) {
            // Si el número es menor que 10 el resultado son sólo las unidades
            textoNumero = textoUnidades;
        } else if (decenas == 1) {
            // Casos especiales del 10 al 19
            switch (numero) {
            case 10: textoNumero = "diez"; break;
            case 11: textoNumero = "once"; break;
            case 12: textoNumero = "doce"; break;
            case 13: textoNumero = "trece"; break;
            case 14: textoNumero = "catorce"; break;
            case 15: textoNumero = "quince"; break;
            case 16: textoNumero = "dieciseis"; break;
            case 17: textoNumero = "diecisiete"; break;
            case 18: textoNumero = "dieciocho"; break;
            case 19: textoNumero = "diecinueve"; break;
            }
        } else if (decenas == 2) {
            // Recoge los casos de la veintena
            if (numero == 20)
                textoNumero = "veinte";
            else
                textoNumero = "veinti" + textoUnidades;
        } else {
            // Recoge los casos desde el 30 hasta el 99
            if (unidades == 0)
                textoNumero = textoDecenas;
            else
                textoNumero = textoDecenas + " y " + textoUnidades;
        }
        // Salida
        System.out.println(textoNumero);
    }
}
```

MAYÚS + ALT + F: NetBeans formatea el código de forma estándar generando saltos de línea, tabulaciones, espacios antes y después de operadores, etc.

Salidas anticipadas: break y continue

- ¿Cuándo conviene utilizar break y continue en Java?

3.3. Salidas anticipadas

Dependiendo de la lógica que implementar en un programa, puede ser interesante terminar un bucle antes de tiempo y no esperar a que termine por su condición (realizando todas las iteraciones). Para poder hacer esto disponemos de:

- `break`: interrumpe completamente la ejecución del bucle.
- `continue`: detiene la iteración actual y continúa con la siguiente.

Cualquier programa puede escribirse sin utilizar `break` ni `continue`; se recomienda evitarlos, ya que rompen la secuencia natural de las instrucciones. Veamos un ejemplo:

```
i = 1;
while (i <= 10) {
   System.out.println("La i vale" + i);
   if (i == 2) {
      break;
   }
   i++;
}
```

En un primer vistazo da la impresión de que el bucle ejecutará 10 iteraciones, pero cuando está realizando la segunda (`i` vale 2), la condición de `if` se evalúa como cierta y entra en juego `break`, que interrumpe completamente el bucle, sin que se ejecuten las sentencias restantes de la iteración en curso ni el resto de las iteraciones. Tan solo se ejecutan dos iteraciones y se obtiene:

```
La i vale 1
La i vale 2
```

Veamos otro ejemplo:

```
i = 0;
while (i < 10) {
   i++;
   if (i % 2 == 0) { //si i es par
      continue;
   }
   System.out.println("La i vale " + i);
}
```

Cuando la condición `i % 2 == 0` sea cierta, es decir, cuando `i` es par, la sentencia `continue` detiene la iteración actual y continúa con la siguiente, saltándose el resto del bloque de instrucciones. `System.out.println` solo llegará a ejecutarse cuando `i` sea impar, o dicho de otro modo: en iteraciones alternas. Se obtiene la salida por consola:

```
La i vale 1
La i vale 3
La i vale 5
La i vale 7
La i vale 9
```

Bucles anidados

3.4. Bucles anidados

En el uso de los bucles es muy frecuente la anidación, que consiste en incluir un bucle dentro de otro, como describe la Figura 3.4.

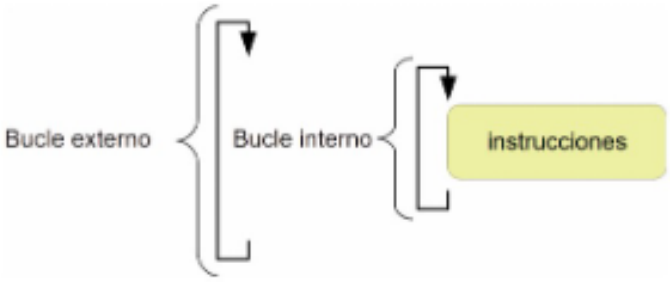

Figura 3.4. *Bucles anidados.*

Al hacer esto se multiplica el número de veces que se ejecuta el bloque de instrucciones de los bucles internos. Los bucles anidados pueden encontrarse relacionados cuando las variables de los bucles más externos intervienen en el control de la iteración de un bucle interno; o independientes, cuando no existe relación alguna entre ellos.

3.4.1. Bucles independientes

Cuando los bucles anidados no dependen, en absoluto, unos de otros para determinar el número de iteraciones, se denominan *bucles anidados independientes*. Veamos un ejemplo sencillo, la anidación de dos bucles for:

```
for (i = 1; i <= 4; i++) {
   for (j = 1; j <= 3; j++) {
      System.out.println("Ejecutando...");
   }
}
```

El bucle externo, controlado por la variable `i`, realizará cuatro iteraciones, donde `i` toma los valores 1, 2, 3 y 4. En cada una de ellas, el bucle interno, controlado por `j`, realizará tres iteraciones, tomando `j` los valores 1, 2 y 3. En total, el bloque de instrucciones se ejecutará doce veces.

Anidar bucles es una herramienta que facilita el procesado de tablas multidimensionales (Unidad 5). Se utiliza cada nivel de anidación para manejar el índice de cada dimensión. Sin embargo, el uso descuidado de bucles anidados puede convertir un algoritmo en algo ineficiente, disparando el número de instrucciones ejecutadas.

E0314. Diseñar una aplicación que muestre las tablas de multiplicar del 1 al 10.

```
public static void main(String[] args) {
    // Proceso & Salida
    for (int i = 1; i <= 10; i++) {
        System.out.println("Tabla de multiplicar del " + i);
        for (int j = 1; j <= 10; j++) {
            System.out.println(i + " * " + j + " = " + i * j);
        }
    }
}
```

3.4.2. Bucles dependientes

Puede darse el caso de que el número de iteraciones de un bucle interno no sea independiente de la ejecución de los bucles exteriores, y dependa de sus variables de control.

Decimos entonces que son *bucles anidados dependientes*. Veamos el siguiente fragmento de código, a modo de ejemplo, donde la variable utilizada en el bucle externo (`i`) se compara con la variable (`j`) que controla el bucle más interno. En algunas ocasiones, la dependencia de los bucles no se aprecia de forma tan clara como en el ejemplo:

```
for (i = 1; i <= 3; i++) {
   System.out.println("Bucle externo, i=" +i);
   j = 1;
   while (j <= i) {
      System.out.println("...Bucle interno, j=" + j);
      j++;
   }
}
```

que proporciona la salida:

Bucle externo, i=1

...Bucle interno, j=1

Bucle externo, i=2

...Bucle interno, j=1

...Bucle interno, j=2

Bucle externo, i=3

...Bucle interno, j=1

...Bucle interno, j=2

...Bucle interno, j=3

- Durante la primera iteración del bucle `i`, el bucle interno realiza una sola iteración.
- En la segunda iteración del bucle externo, con `i` igual a 2, el bucle interno realiza dos iteraciones.
- En la última vuelta, cuando `i` vale 3, el bucle interno se ejecuta tres veces.

La variable `i` controla el número de iteraciones del bucle interno y resulta un total de 1 + 2 + 3 = 6 iteraciones. Los posibles cambios en el número de iteraciones de estos bucles hacen que, *a priori*, no siempre sea tan fácil conocer el número total de iteraciones.

E0315. Pedir por consola un número n y dibujar un triángulo rectángulo de n elementos de lado, utilizando para ello asteriscos (*). Por ejemplo, para n = 4:

```
* * * *
* * *
* *
*
```

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    int numero;

    // Entrada de datos
    Scanner sc = new Scanner(System.in);
    System.out.println("Escribe un número: ");
    numero = sc.nextInt();

    // Proceso & Salida
    for (int i = numero; i >= 1; i--) {
        for (int j = 1; j <= i; j++) {
            System.out.print("* ");
        }
        System.out.println("");
    }
}
```