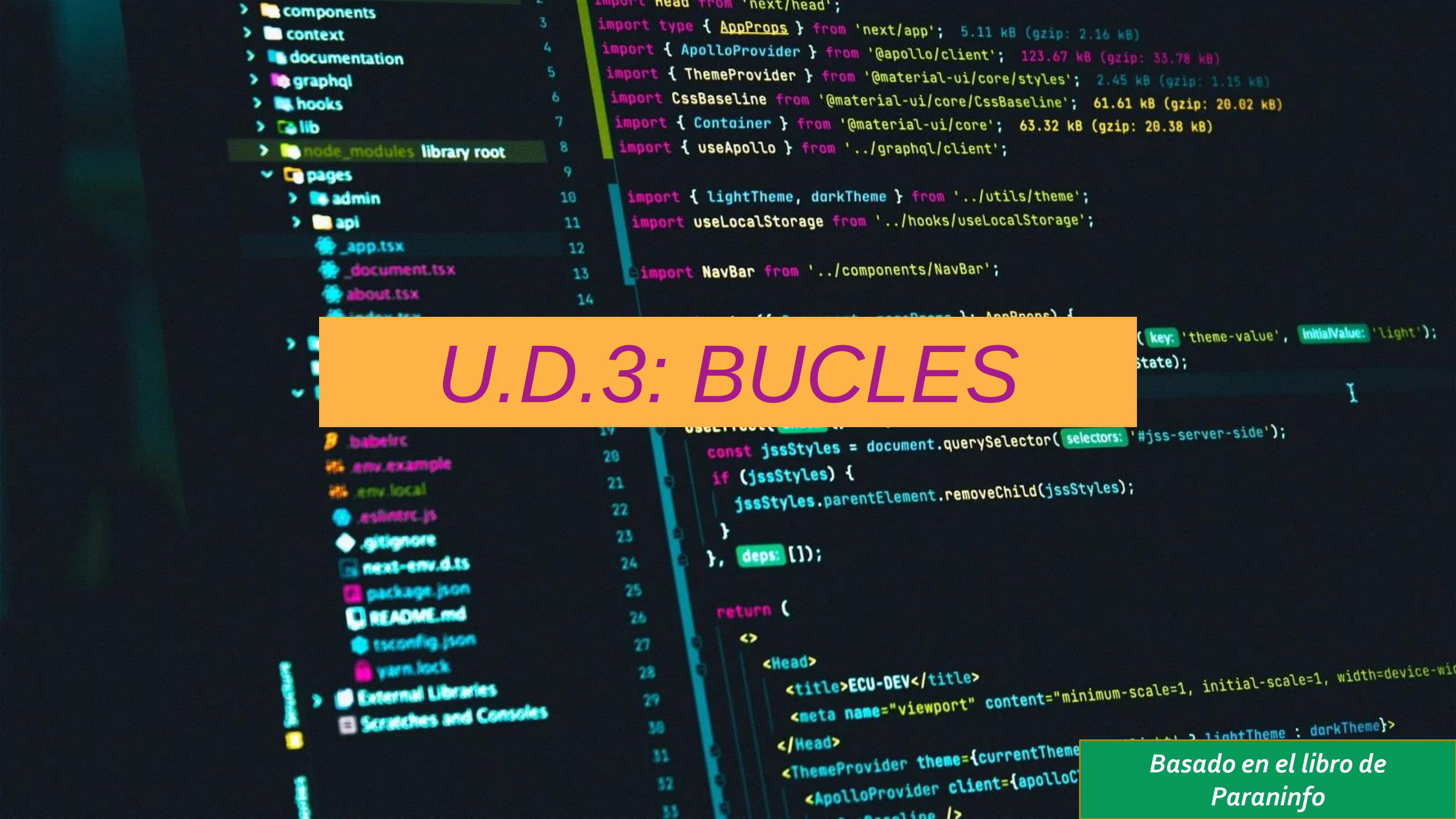


U.D.3: BUCLES

Basado en el libro de Paraninfo

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****.*

INTRODUCCIÓN

El uso de bucles simplifica la estructura 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**.

BUCLES CONTROLADOS POR CONDICIÓN

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

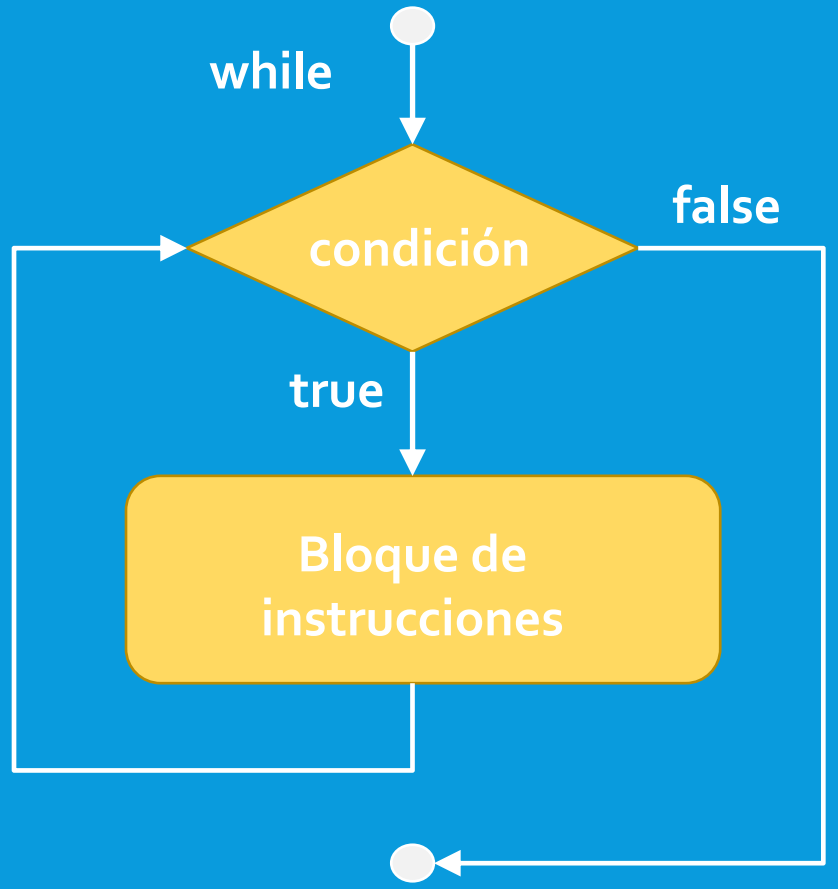
BUCLES CONTROLADOS POR CONDICIÓN: While

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
    …
}
```

BUCLES CONTROLADOS POR CONDICIÓN: While

El comportamiento de este bucle es el siguiente:



BUCLES CONTROLADOS POR CONDICIÓN: While

El comportamiento de este bucle es el siguiente:

1. Se *evalúa* la 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*.

BUCLES CONTROLADOS POR CONDICIÓN: While

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 intCont = 1; // valor inicial

while (intCont <= 3) { // el bucle iterará mientras i sea menor o igual que 3

    System.out.println(intCont); // mostramos i

    intCont++; // incrementamos i
}
```

BUCLES CONTROLADOS POR CONDICIÓN: While

Veamos como se ejecuta:

1. Se declara la variable ***intCont*** y se le asigna el *valor 1*.
2. La instrucción ***while*** evalúa la ***condición (intCont <= 3):*** ¿es 1 ≤ 3? Cierto.
3. Se ejecuta el bucle de instrucciones: ***System.out.println*** e ***intCont++***. Ahora la ***intCont vale 2***.
4. La instrucción ***while*** vuelve a evaluar la condición: ¿es 2 ≤ 3? Cierto.

BUCLES CONTROLADOS POR CONDICIÓN: While

5. Se ejecuta el bucle de instrucciones: ***System.out.println*** e ***intCont++***. Ahora la ***intCont vale 3***.
6. La instrucción ***while*** vuelve a evaluar la condición: ¿es 3 ≤ 3? Cierto.
7. Se ejecuta el bucle de instrucciones: ***System.out.println*** e ***intCont ++***. Ahora la ***intCont 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*.

BUCLES CONTROLADOS POR CONDICIÓN: 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***.

BUCLES CONTROLADOS POR CONDICIÓN: While

Veamos un ejemplo de un bucle ***while*** que nunca llega a ejecutarse.

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

BUCLES CONTROLADOS POR CONDICIÓN: While

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

BUCLES CONTROLADOS POR CONDICIÓN: While

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.

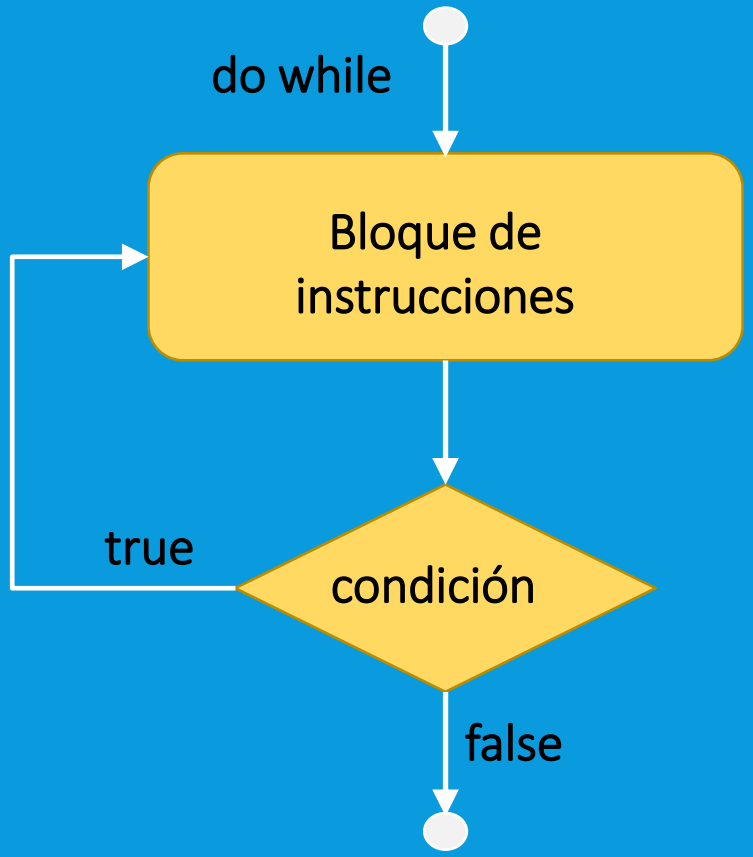
BUCLES CONTROLADOS POR CONDICIÓN: do-while

Disponemos de un segundo bucle controlado por una condición: el bucle ***do – 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);
```

BUCLES CONTROLADOS POR CONDICIÓN: do-while

El comportamiento de este bucle es el siguiente:



BUCLES CONTROLADOS POR CONDICIÓN: do-while

Como hemos visto en el diagrama anterior, su comportamiento sigue los siguientes puntos:

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

BUCLES CONTROLADOS POR CONDICIÓN: do-while

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

```
int intNum = 1;

do {

    System.out.println(intNum);

    intNum++;

} while (intNum <= 10);
```

BUCLES CONTROLADOS POR CONDICIÓN: do-while

Debemos recordar que es el único bucle que termina en punto y coma (;). Mientras el bucle ***while*** se puede ejecutar de 0 a infinitas veces, el ***do - while*** lo hace de 1 a infinitas veces. De hecho, la única diferencia con el bucle ***while*** es que ***do – while*** se ejecuta, al menos, una vez.

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
    …
}
```

BUCLES CONTROLADOS POR CONTADOR: for

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.

***BUCLES CONTROLADOS POR CONTADOR*: for**

Donde,

- ***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***.

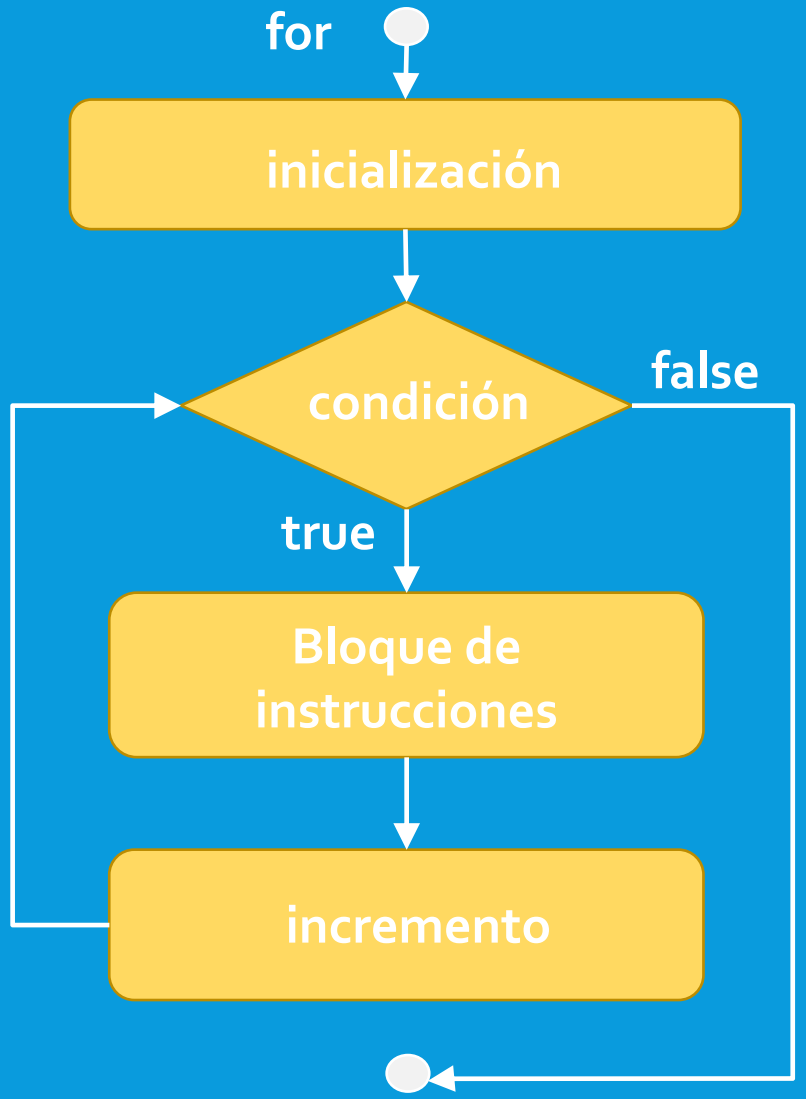
***BUCLES CONTROLADOS POR CONTADOR*: for**

Donde,

- ***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.

BUCLES CONTROLADOS POR CONTADOR: for

Comportamiento:



BUCLES CONTROLADOS POR CONTADOR*: *for

El funcionamiento de ***for*** descrito en la transparencia anterior, sigue 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.

BUCLES CONTROLADOS POR CONTADOR*: *for

3. Cuando termina de ejecutarse el bloque de instrucciones, se ejecuta el incremento.

4. Se vuelve de nuevo al punto 2 (evaluación de la condición).

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.

BUCLES CONTROLADOS POR CONTADOR: for

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

```
for (int intNum = 1; intNum <= 2; intNum++)
   System.out.println ("intNum vale " + intNum);
```

BUCLES CONTROLADOS POR CONTADOR: for

En este caso, la variable ***intNum***, además de iniciarse, también se declara en la zona de inicialización. Esto significa que ***intNum*** solo puede usarse dentro de la estructura for.

***BUCLES CONTROLADOS POR CONTADOR*: for**

A continuación se muestra la ejecución del bucle anterior paso a paso:

1. Primero se ejecuta la inicialización: ***intNum* = 1**;
2. Evaluamos la condición: ¿es cierto que ***intNum* ≤ 2**?. Es decir: ¿**1 ≤ 2**?
3. Cierto. Ejecutamos el bloque de instrucción: ***System.out.println(…)***
4. Obtenemos el mensaje: << ***intNum vale 1***>>.
5. Terminado el bloque de instrucciones, ejecutamos el incremento: ***(intNum ++) intNum vale 2***.

BUCLES CONTROLADOS POR CONTADOR: for

6. Evaluamos la condición: ¿es cierto que ***intNum* ≤ 2**?. Es decir: ¿**2 ≤ 2**?
7. Cierto. Ejecutamos el bloque de instrucción: ***System.out.println(…)***
8. Obtenemos el mensaje: << ***intNum vale 2***>>.
9. Ejecutamos el incremento: ***(intNum ++) i intNum vale 3***.
10. Evaluamos la condición: ¿es cierto que ***intNum* ≤ 2**?. Es decir: ¿**3 ≤ 2**?
11. Falso. El bucle termina y continua la ejecución de las sentencias que siguen a la estructura ***for***.

SALIDAS ANTICIPADAS

A continuación se muestra la Dependiendo de la lógica a 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.

SALIDAS ANTICIPADAS

A continuación se muestra la Dependiendo de la lógica a 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.

SALIDAS ANTICIPADAS

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

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

SALIDAS ANTICIPADAS

En un primer vistazo da la impresión de que el bucle ejecutará 10 iteraciones, pero cuando está realizando la segunda ***(intNum vale 2),*** la condición ***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:

intNum vale 1

intNum vale 2

SALIDAS ANTICIPADAS

Veamos otro ejemplo:

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

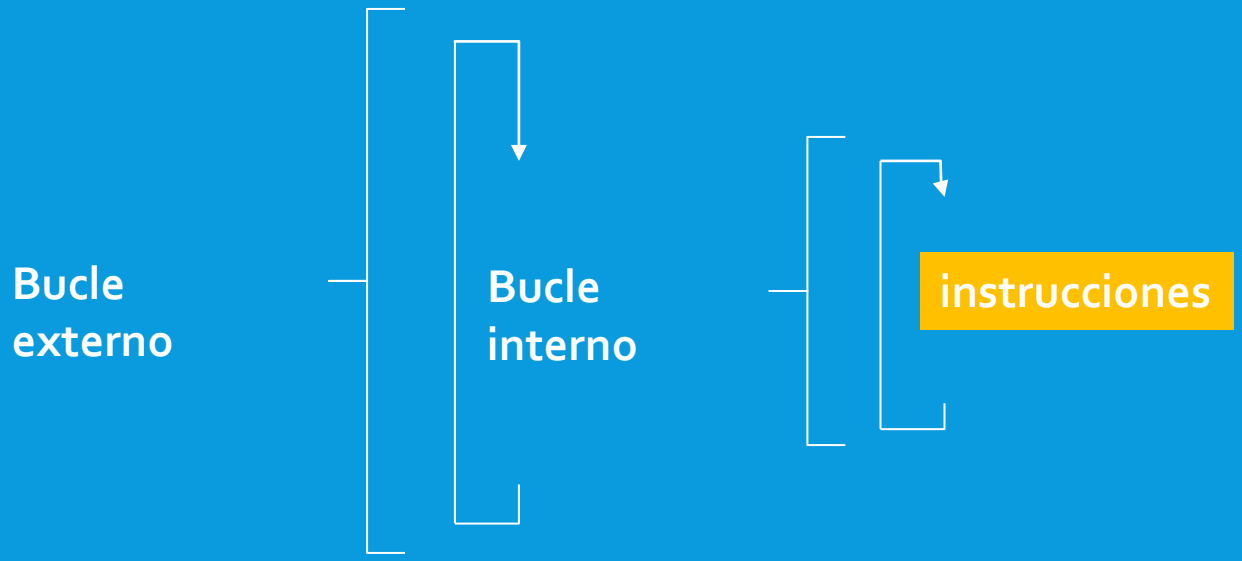
SALIDAS ANTICIPADAS

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

intNum vale 1
intNum vale 3
intNum vale 5
intNum vale 7
intNum vale 9

BUCLES ANIDADOS

En el uso de los bucles es muy frecuente la anidación, que consiste en incluir un bucle dentro de otro, como se ve en la siguiente figura:



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 independiente, cuando no existe relación alguna entre ellos.

BUCLES ANIDADOS. 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 en un ejemplo sencillo, la anidación de dos bucles for:

```
for ( int intNum = 1; intNum <= 4; intNum++)
   for (int intNum2 = 1; intNum2 <= 3; intNum2++)
      System.out.println ("Ejecutando…");
```

BUCLES ANIDADOS. ***Bucles independientes***

El bucle externo, controlado por la variable ***intNum***, realizará cuatro iteraciones, donde ***intNum*** toma los valores 1, 2, 3 y 4. En cada una de ellas, el bucle interno, controlado por ***intNum2***, realizará tres iteraciones, tomando ***intNum2*** los valores 1, 2 y 3. En total, el bloque de instrucciones se ejecutará doce veces.

BUCLES ANIDADOS**.** ***Bucles independientes***

Anidar bucles es una herramienta que facilita el procesado de tablas multidimensionales. 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.

BUCLES ANIDADOS*. *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 (***intNum***) se compara con la variable (***intNum2***) que controla el bucle más interno.

BUCLES ANIDADOS. Bucles dependientes

En algunas ocasiones, la dependencia de los bucles no se aprecia de forma tan clara como en el ejemplo:

```
int intNum2;
for (int intNum = 1; intNum <= 3; intNum++) {
   System.out.println ("Bucle externo, intNum= " + intNum);
   intNum2 = 1;
   while (intNum2 <= intNum) {
      System.out.println ("… Bucle interno, intNum2= " + intNum2);
      intNum2++;
   }
}
```

BUCLES ANIDADOS. *Bucles dependientes*

que proporciona la salida:

Bucle externo, *intNum* =1
… Bucle interno, *intNum2* =1
Bucle externo, *intNum* =2
… Bucle interno, *intNum2* =1
… Bucle interno, *intNum2* =2
Bucle externo, *intNum* =3
… Bucle interno, *intNum2* =1
… Bucle interno, *intNum2* =2
… Bucle interno, *intNum2* =3

BUCLES ANIDADOS. ***Bucles dependientes***

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

BUCLES ANIDADOS*. *Bucles dependientes

La variable ***intNum*** 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.