Universidad Europea

3

Sentencias de repetición

Ve más allá

CONTENIDOS

- **Unidad 2: Sentencias de Control**
  1. Expresiones condicionales
  2. Sentencias de decisión
  3. **Sentencias de repetición**

Universidad Europea

ÍNDICE

- Sentencias de decisión
  1. Introducción
  2. Sentencia while
     - Ejercicios
     - Errores típicos
  3. Sentencia do-while
     - Ejercicios
     - Errores típicos
  4. Sentencia for
     - Ejercicios
  5. Sentencia break
  6. Sentencia continue
  7. Sentencia return

Universidad Europea

Introducción

Estructuras o sentencias de iteración, que permiten ejecutar secciones específicas del código una cantidad determinada de veces.

Iterar es repetir una acción varias veces

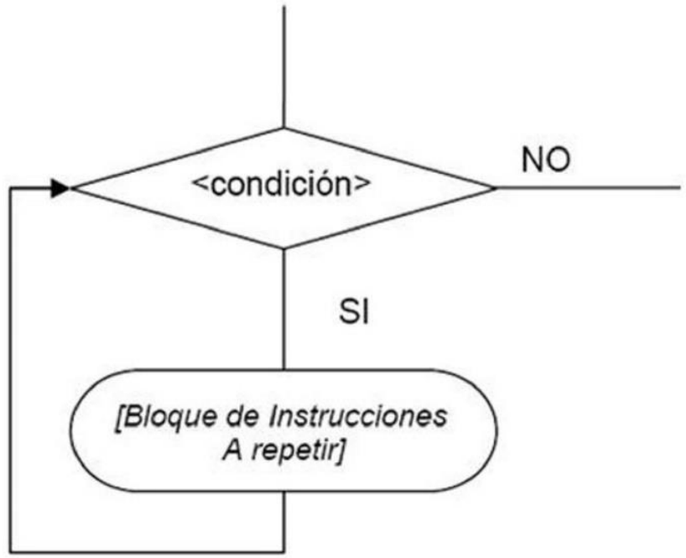
En programación nos va a interesar iterar sobre un trozo de código un número determinado de veces, evitando copiarlo n veces (en muchos casos un nº de veces variable)

A este tipo de sentencias las conocemos tradicionalmente como bucles.

Sentencia while

Comprueba la condición y ejecuta el bloque mientras la condición sea cierta

El diagrama de flujo de esta instrucción es el siguiente:

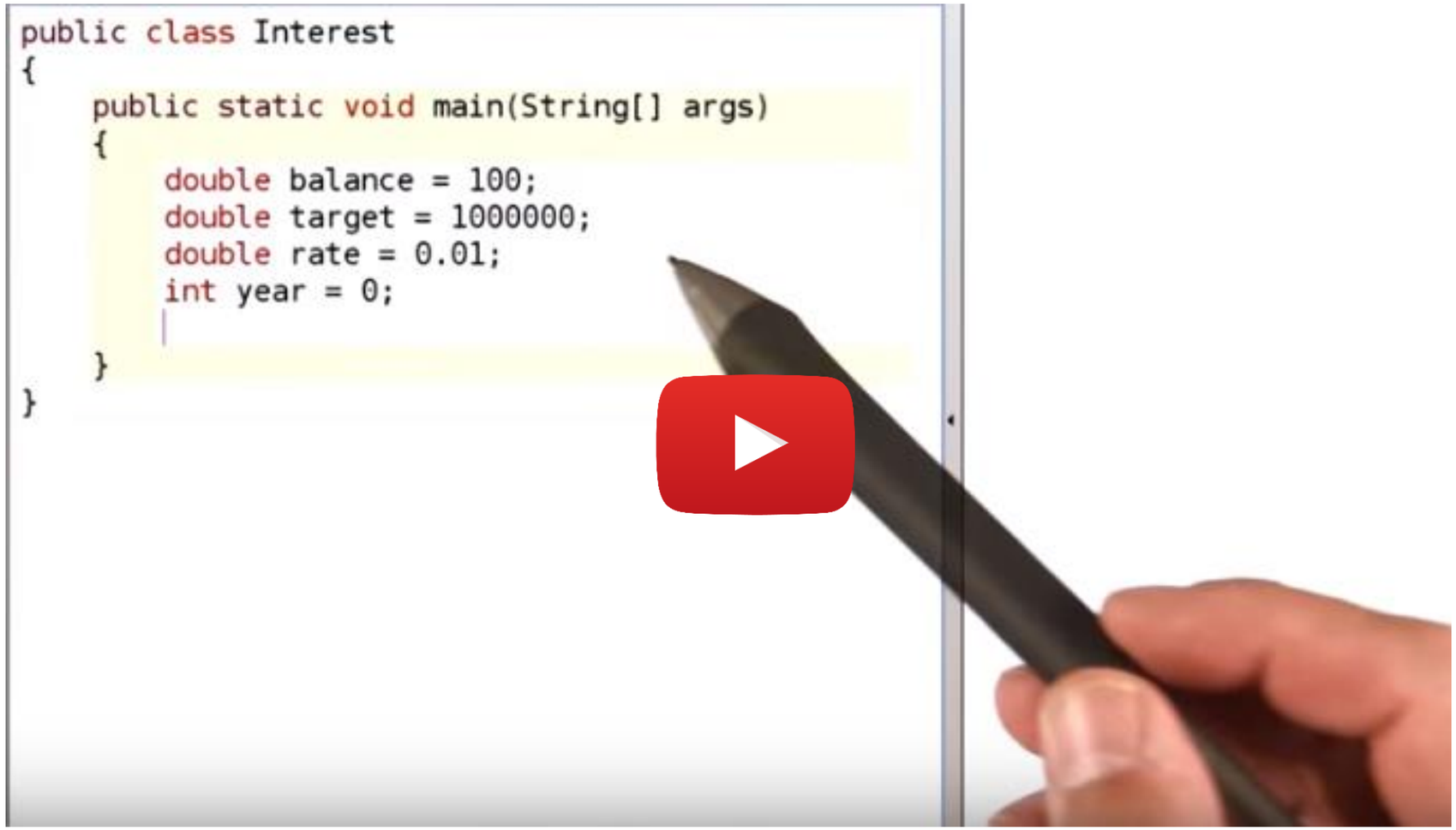


Y la sintaxis en java es la siguiente:

while

```
while(condición){
    cuerpo
}
```

Sentencia while



Sentencia while

```
// TODO: Change the program so that it will calculate the number of years
// to reach 1 billion dollars instead of 1 million.
public class InterestCalculator
{
    public static void main(String[] args)
    {
        double balance = 100;
        int year = 0;

        while (balance < 1000000)
        {
            double interest = balance * .01;
            balance = balance + interest;
            year++;
        }
        System.out.println("It will take " + year + " years.");
    }
}
```

 <- solución

Ejercicio 1

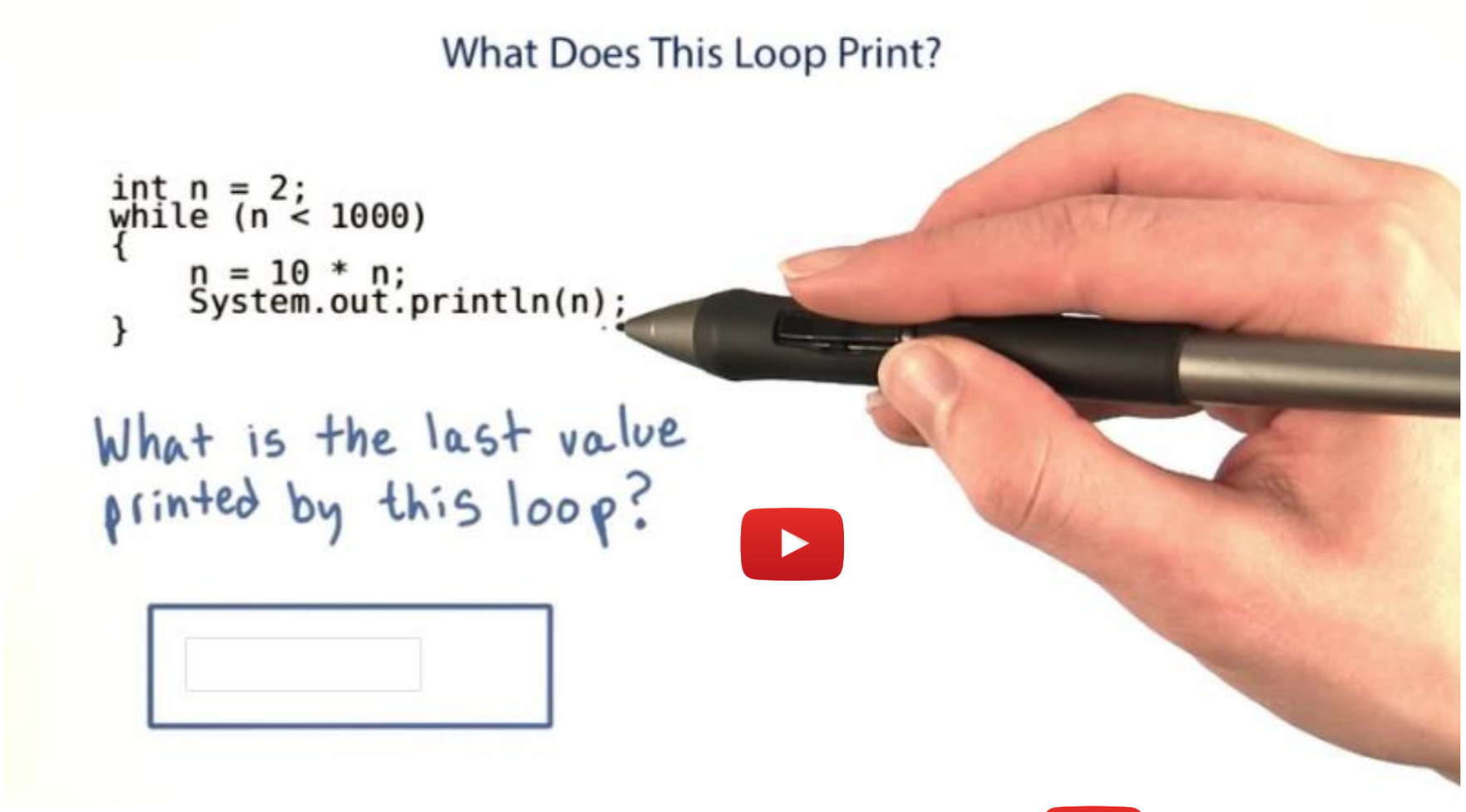
Crea un código que solicite un número y escriba por pantalla todos los números desde dicho número hasta el 0, separados por un espacio

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Introduzca un número: ");
    int numero = in.nextInt();

    while (...) {
        System.out.print(... + " ");
    }
}
```

¿Qué pondrías en los ...?

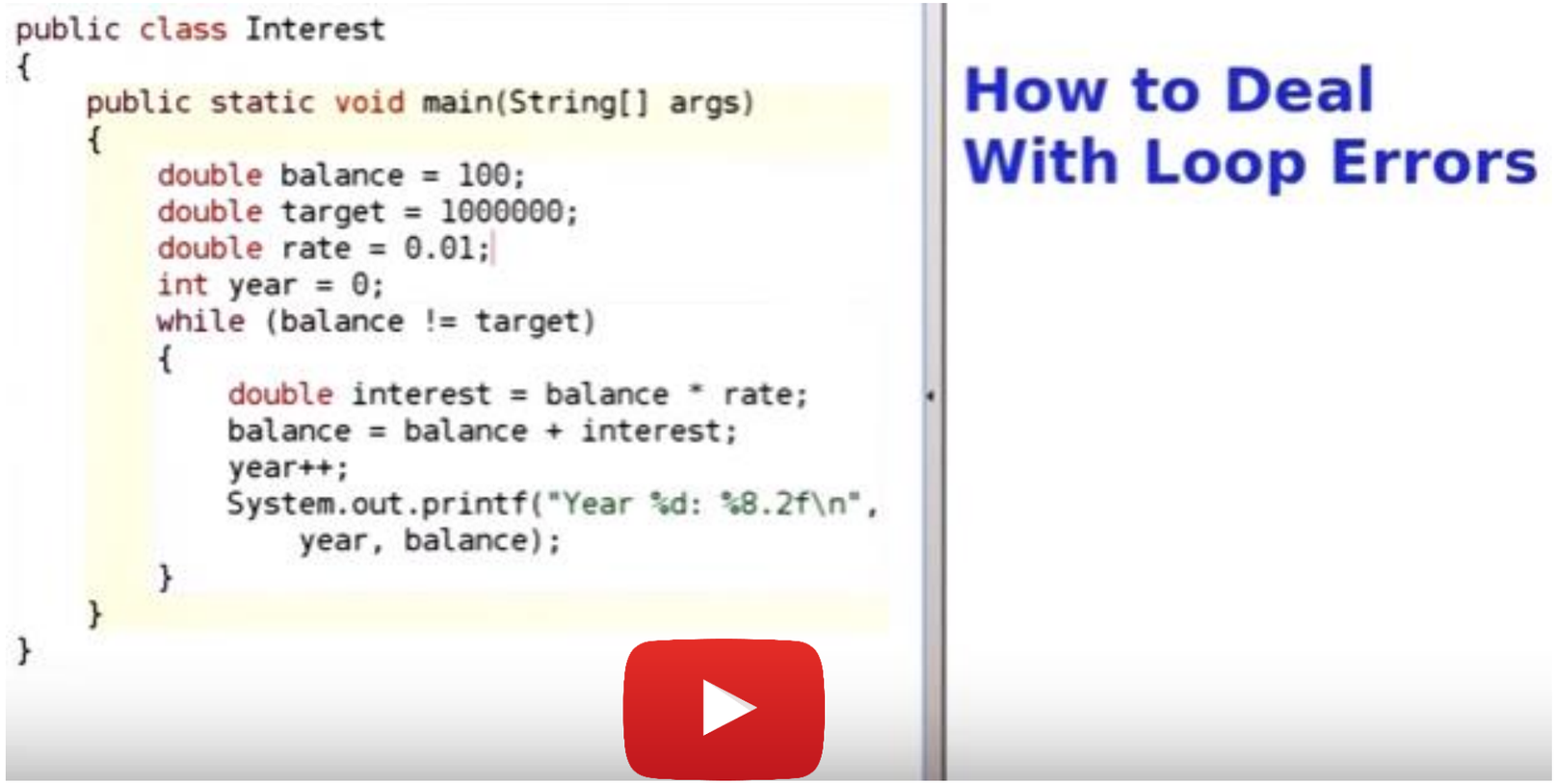
a. numero > 0 y numero--
b. numero >= 0 y numero++
c. numero <= 0 y numero--
d. numero >= 0 y numero--

Ejercicio 2



<- solución

Sentencia while (Errores típicos)



Sentencia while (Errores típicos)

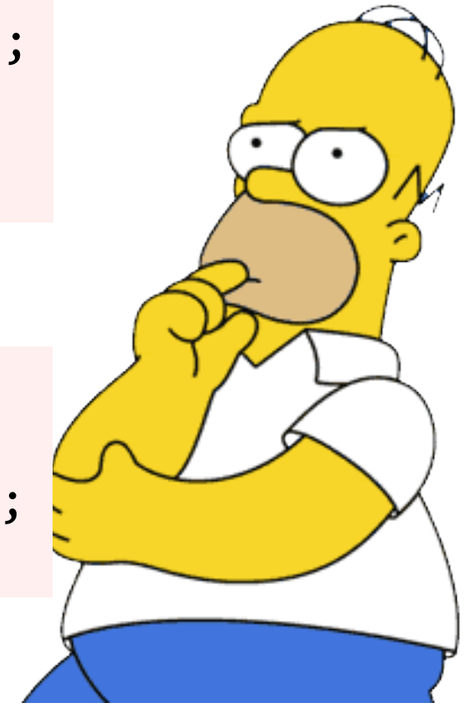
```
// TODO: Fix the error so that the loop stops
// when the balance has reached the target

import java.util.Scanner;

public class Interest
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Target: ");
        double balance = 100;
        double target = in.nextDouble();
        double rate = 0.01;
        int year = 0;
        while (balance != target)
        {
            double interest = balance * rate;
            balance = balance + interest;
            year++;
            System.out.printf("Year %d: %8.2f\n", year, balance);
        }
    }
}
```

<- solución

Sentencia while (Errores típicos)

```
int x = 0;
while (x<10)  {
      System.out.println(x);
      x++;
}
```

```
int x = 0;
while (x>=0)  {
      System.out.println(x);
      x++;
}
```



```
int x = 0;
while (x<10)  {
      System.out.println(x);
}
```

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

Sentencia while (Errores típicos)

En general en las iteraciones se pueden dar situaciones no deseadas:

Código muerto: Si no se ejecuta nunca, es decir, si la condición de entrada siempre es false, el bloque no se ejecutará nunca. Esto puede suceder porque las variables incluidas en la condición no estén bien inicializadas o porque hayamos escrito una condición incorrecta

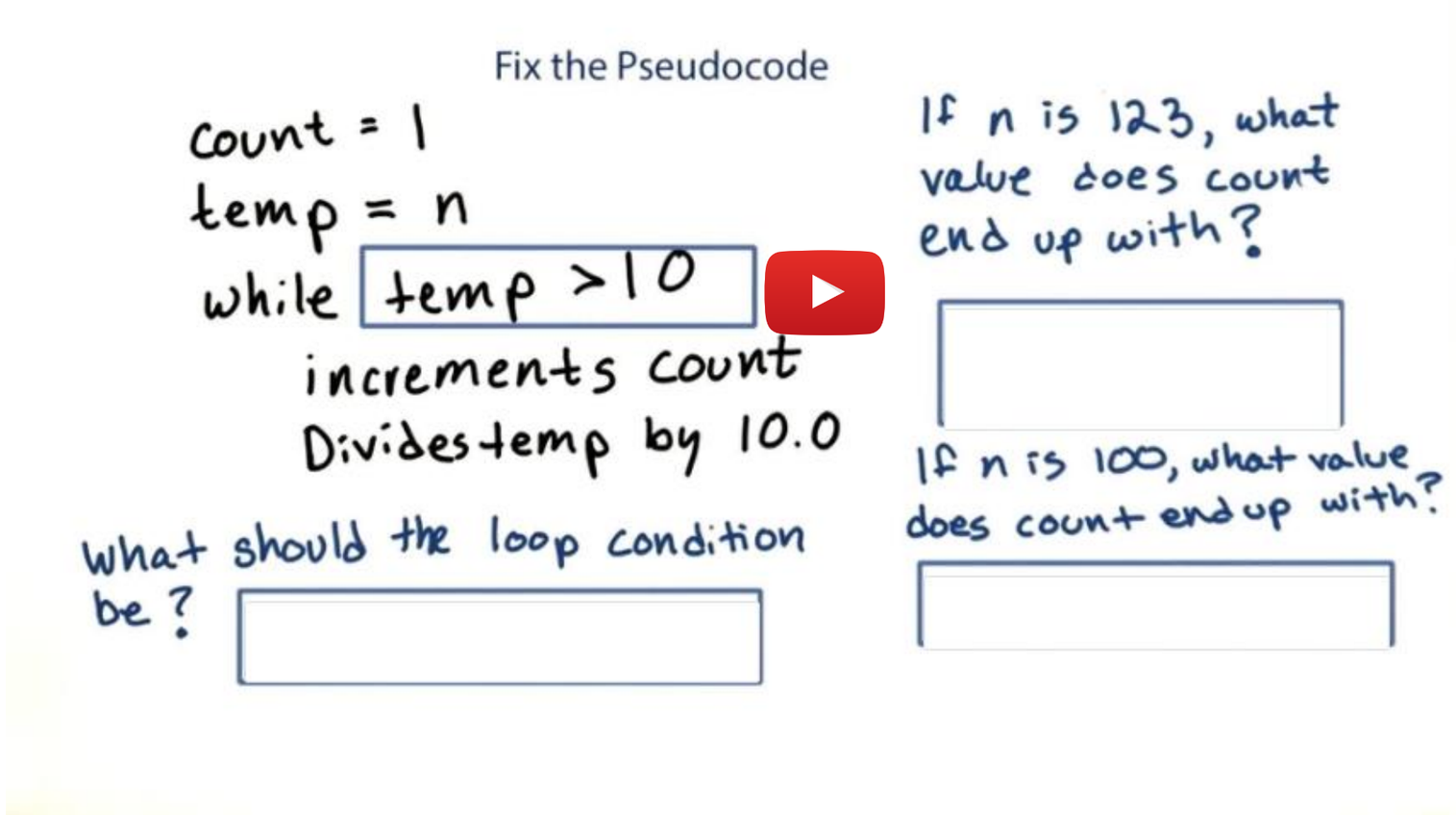
Bucle infinito: Si nunca deja de ejecutarse, es decir, si la condición es true y nunca cambia a false. Esto puede suceder, si dentro del bloque no se modifica la condición.

Sentencia while (Errores típicos)

Trace the Buggy Code


```
int balance = 100;
int target = 200;
rate = .1;
year = 0;
while (balance >=  target)
{
    double interest = balance * rate;
    balance = balance + interest;
    year++;
    System.out.printf("Year %d: %8.2f", year, balance);
}
```

Sentencia while (Errores típicos)

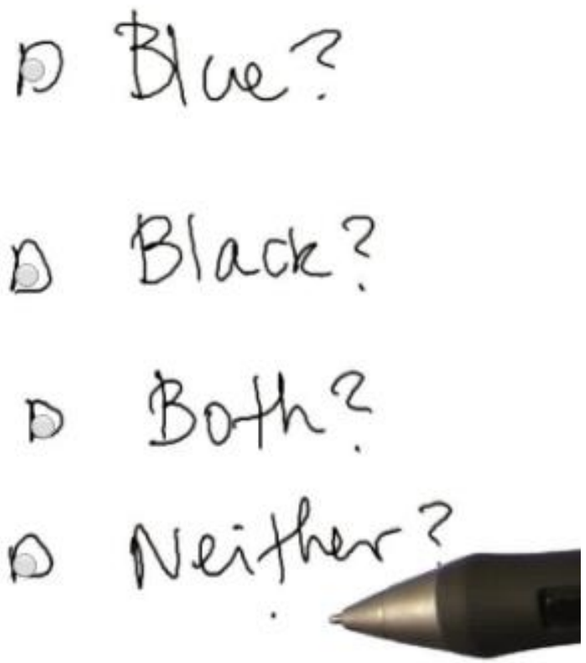


<- solución

Sentencia do-while

```
int value = 100;
while (value >= 100)
{
   System.out.print("Enter a value < 100: ");
   value = in.nextInt();
}
```

```
System.out.print("Enter a value < 100: ");
int value = in.nextInt();
while (value >= 100)
{
   System.out.print("Enter a value < 100: ");
   value = in.nextInt();
}
```



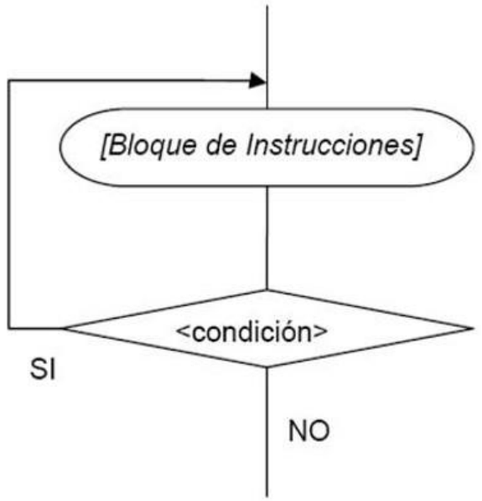
<- solución

Universidad Europea

Sentencia do-while

Bucle de tipo posprueba, es decir, que primero se ejecuta el bloque de código y luego se pregunta si hay que volver a repetirlo.

El diagrama de flujo de esta instrucción es el siguiente:



Y la sintaxis en java es la siguiente:

do-while

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

Sentencia do-while

```
while (value >= 100)
{
    System.out.print("Enter a value < 100: ");
    value = in.nextInt();
}
```

Ejercicio

Realiza un programa que solicite un número entre 1 y 10, y repita la pregunta si el usuario introduce un número que no esté en dicho rango de valores.

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int numero;
    do {
        System.out.print("Introduzca un número [1-10]: ");
        numero = in.nextInt();
    } while (...);
    System.out.println("Has introducido el número: " + numero);
}
```

¿Cómo lo completarías?

a) numero < 1 || numero > 10
b) numero < 1 && numero > 10
c) numero >= 1 || numero <= 10
d) numero >= 1 && numero <= 10

Sentencia do-while (Errores típicos)

```
int x = 0;
do  {
      System.out.println(x);
      x++;
} while (x<10);
```

```
int x = 0;
do  {
      System.out.println(x);
      x++;
} while (x<0);
```

```
int x = 0;
do  {
      System.out.println(x);
      x++;
} while (x>0);
```

Sentencia do-while (Errores típicos)

El usuario no siempre va a hacer lo que esperamos/lo que le hemos pedido. Debemos prevenir estas situaciones para evitar errores de funcionamiento.

Por ejemplo, ¿qué problema tiene el siguiente código?

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int numero;

    System.out.print("Introduzca un número entero: ");
    while (!in.hasNextInt()) {
        System.out.println("Debe ser un número entero");
        System.out.print("Introduzca un número entero: ");
        in.nextLine();  /* consume y tira la línea actual */
    }
    numero = in.nextInt();
    System.out.println("Has introducido el número: " + numero);
}
```

Sentencia for

Sentencia for

Esta sentencia permite ejecutar un bloque de código p.e. un número determinado de veces.

Es muy útil cuando sabemos el nº de veces que deseamos ejecutar el bucle.

Es muy cómodo de utilizar ya que condensa en la misma línea toda la información para saber cómo se va a comportar el bucle

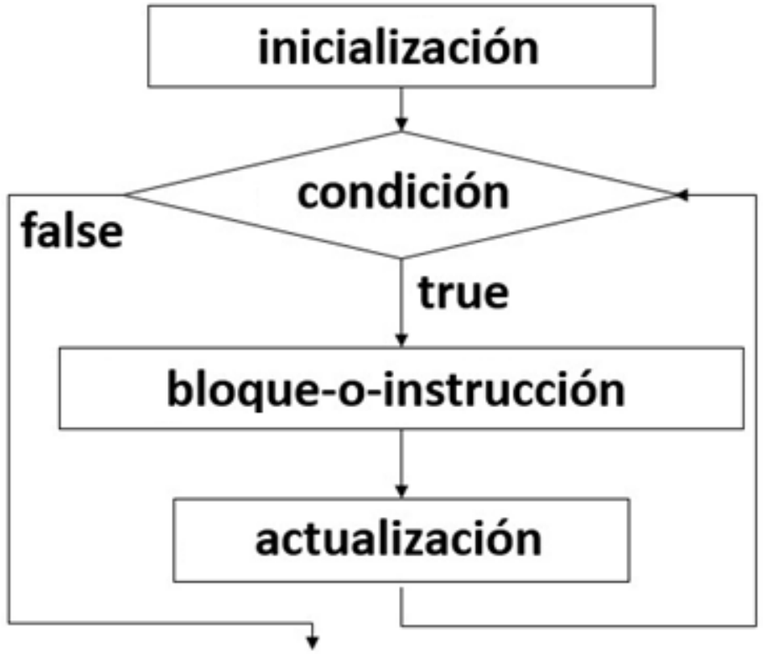
Esta es su sintaxis en Java:

for

```
for(inicialización; condición; iteración){
    cuerpo
}
```

Sentencia for

Y este su diagrama de flujo:



Siempre equivalente a un bucle while

Como en el caso de do-while, muchas veces un bucle for es más compacto que un while

Ejercicio 1

```
// Actually, Elizas todo list should look like this:
// 1. Eat
// 2. Sleep
// 3. Eat
// 4. Sleep
// 5. Eat
// 6. Sleep
// TODO: Change the code inside the for loop to print out this todo list
public class TodoList
{
    public static void main(String[] args)
    {
        for (int counter = 1; counter <= 6; counter++)
        {
            System.out.println(counter + ". Sleep");
        }
    }
}
```

<- solución

Sentencia for

Se recomienda utilizarlo para recorrer estructuras como Strings y Arrays, que veremos más adelante.

Al condensar información en una línea, no está recomendado cuando cualquiera de los elementos que lo definen (la inicialización, la condición o la actualización) es largo o complejo.

Ejercicio 2

Realiza un programa que cuente del 1 al 100 con un salto definido por el usuario. Si el usuario introduce el 1 mostrará todos los números, pero si el usuario introduce otro, por ejemplo, el 5 mostrará: 1 6 11 16 y así sucesivamente.

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Introduzca un número: ");
    int salto = in.nextInt();
    for (int i = 1; i <= 100; actualizacion) {
        System.out.print(i + " ");
    }
}
```

¿Cómo actualizarías *i* ?

a) i++
b) i+=salto
c) i--
d) i = salto

Ejercicio 3

```
// Make the loop count down the days left until vacation.
public class CountingDown
{
   public static void main(String[] args)
   {
      // TODO: Make this loop count down instead of up.
      // It should print
      //   20 days left
      //   19 days left
      //   18 days left
      // and so on, down to 0 days left.
      for (int i = 0; i <= 10; i++)
      {
         System.out.println(i);
      }
   }
}
```

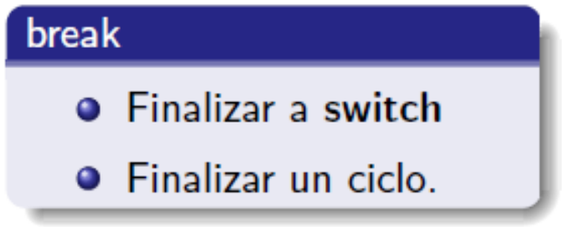
<- solución

Sentencia break

La sentencia break se debe utilizar en la instrucción switch para finalizar una rama (case).

También se puede utilizar para alterar el flujo del bucle, ya que finaliza dicho bucle inmediatamente. Este segundo uso no está aconsejado, ya que se considera programación no estructurada. Entender un programa o corregir errores es más complicado cuando el código las utiliza. Es necesario que la entiendas por si te la encuentras en algún programa que debes modificar.

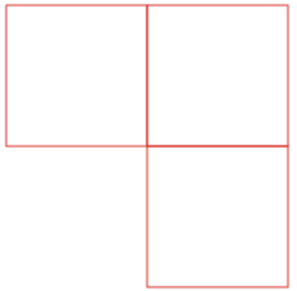
Algunos programadores la utilizan por comodidad o rapidez a la hora de codificar, pero siempre se puede sustituir por una condición más compleja



Sentencia break

Por ejemplo, el siguiente bucle debería ejecutarse 10 veces, desde j = 0 hasta j = 9, sin embargo, la utilización de la sentencia break, rompe la iteración del bucle, de tal manera que tras la primera ejecución el bucle acaba, habiéndose ejecutado una sola vez.

```
for(int j = 0; j<10; j++){
  sentencia 1;
  sentencia 2;
  sentencia 3;
  break;
}
```



Ejercicio

Dado el siguiente programa que tiene la sentencia break, trata de modifícarlo para que realice la misma tarea, pero sin incluir el break.

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Introduzca un número: ");
    int numero = in.nextInt();

    while (true) {
        System.out.print(numero + " ");
        numero--;
        if (numero == 0) {
            break;
        }
    }
}
```

¿Ves el riesgo de un bucle infinito?

Sentencia continue

La sentencia de continue, como el break, es de tipo de control de bucles.

Se puede incluir dentro de cualquiera de los tipos vistos (while, do-while, for), provocando que se ejecute la siguiente iteración del bucle, es decir, ignora las sentencias posteriores al “continue”.

Al igual que la sentencia break en los bucles, está desaconsejada ya que añade complejidad al código, haciéndolo más difícil de entender.

continue

La sentencia **continue** forza en un *ciclo* la ejecución de una nueva *iteración* descartando el procesamiento de la *iteración* actual.

Sentencia continue

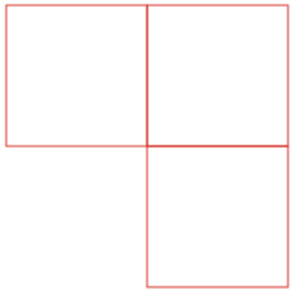
Este bucle se ejecuta 10 veces, pero con la salvedad de que la sentencia 4 no se ejecuta ninguna vez. Es decir, se ejecutan las sentencias 1, 2 y 3 y cuando se llega a la sentencia de control continue se vuelve de nuevo a comprobar la condición del for y en caso de cumplirse de nuevo a la sentencia 1 y así sucesivamente.

```
for(int j = 0; j<10; j++){
    sentencia 1;
    sentencia 2;
    sentencia 3;
    continue;
    sentencia 4;
 };
```

Ejercicio

Dado el siguiente programa que tiene la sentencia continue, modifícalo para que realice la misma tarea, pero sin incluir el continue.

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Introduzca un número: ");
    int numero = in.nextInt();

    while (numero-- > 0) {
        if (numero % 2 != 0) {
            continue;
        }
        System.out.print(numero + " ");
    }
}
```



Sentencia return

La sentencia return se utiliza para terminar un método o función y opcionalmente devolver un valor al método de llamada.

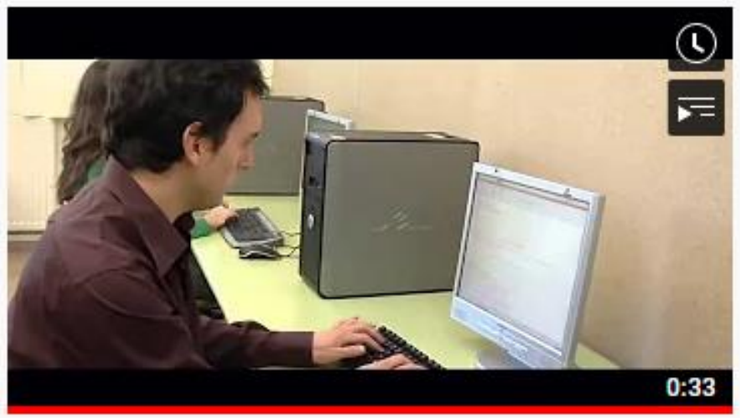
En el código de una función siempre hay que ser consecuentes con la declaración que se haya hecho de ella. Por ejemplo, si se declara una función para que devuelva un entero, es imprescindible colocar un return final para salir de esa función, independientemente de que haya otros en medio del código que también provoquen la salida de la función.

Si el valor a retornar es void, se puede omitir ese valor de retorno, con lo que la sintaxis se queda en un sencillo:

```
return;
```

y se usaría simplemente para finalizar el método o función en que se encuentra, y devolver el control al método o función de llamada

APUNTE...



Universidad Europea

GRACIAS

Pedro J. Camacho

Universidadeuropea.com

Ve más allá