Estructuras de control en Java

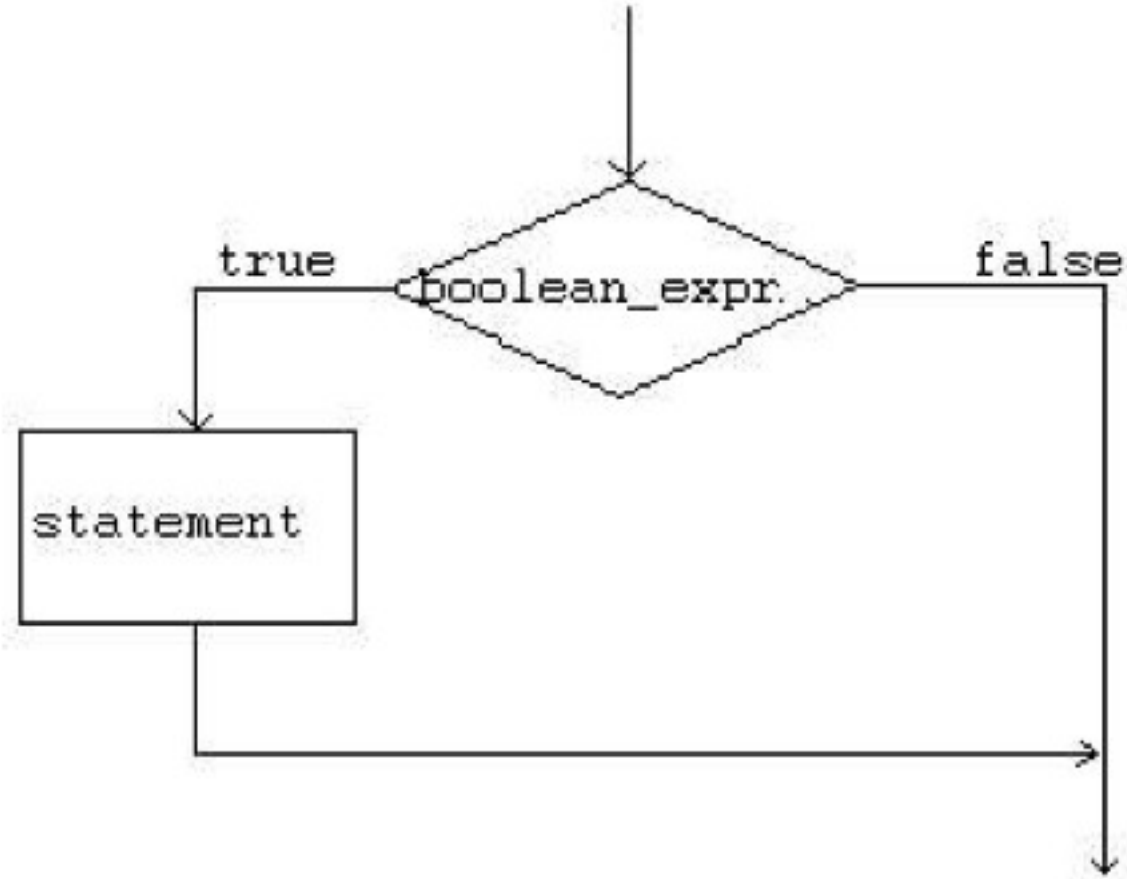
Estructuras de control

- Estructuras de control
  - Nos permite cambiar el orden de las declaraciones ejecutadas en nuestros programas.
- Hay dos tipos de estructuras de control
  - Estructuras de selección / decision control structures
    - Nos permite seleccionar secciones específicas del código para ser ejecutado, a partir de una condición.
  - Estructuras de iteración / repetition control structures
    - Nos permite ejecutar secciones específicas del código una cantidad determinada de veces.

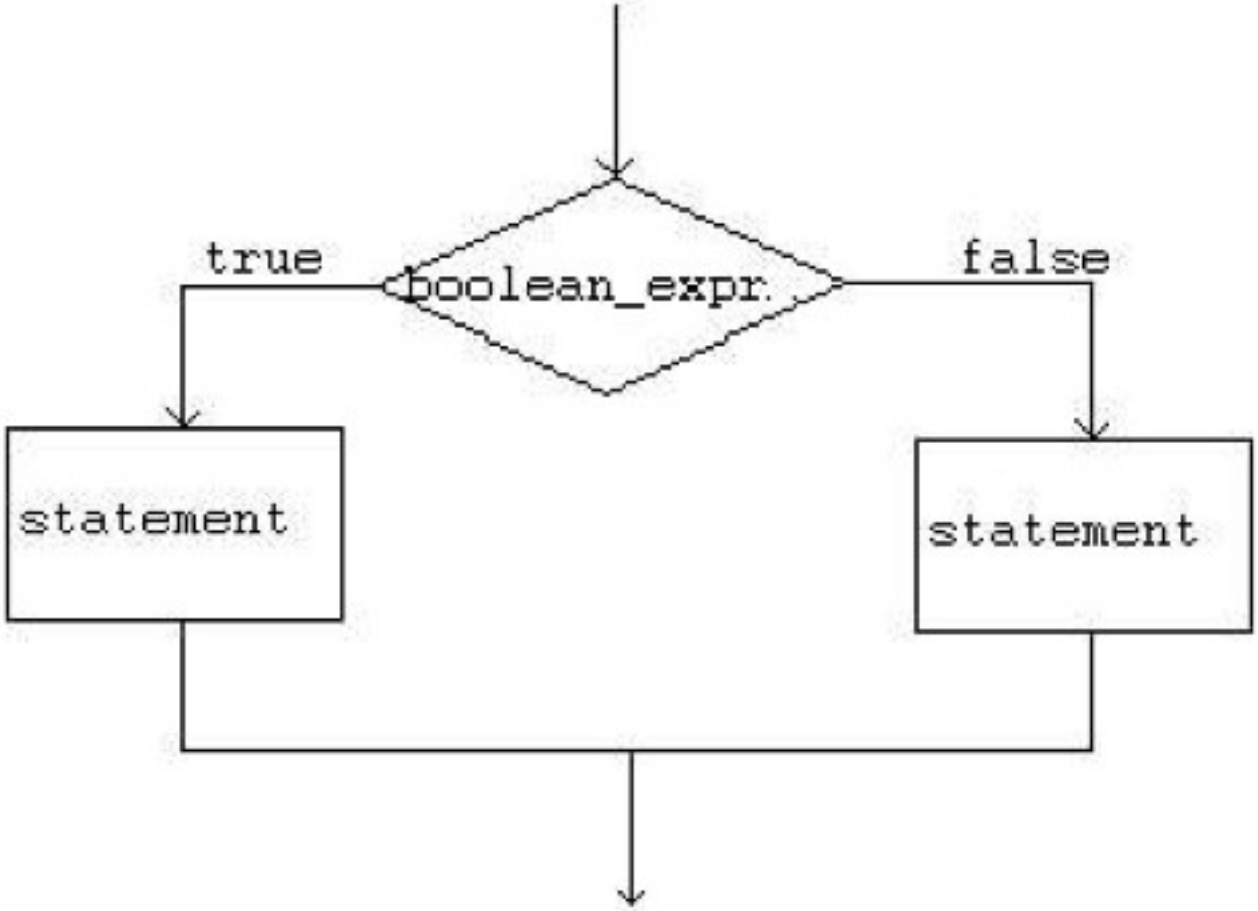
Estructuras de selección

- Estructuras de selección
  - Declaraciones que nos permiten seleccionar y ejecutar bloques específicos del código mientras otras partes son ignoradas.
- Tipos:
  -
```
if( boolean_expression )
        statement;
```
  -
```
if( boolean_expression ){
        statement1;
    }
    else{
        statement2;
    }
```
  -
```
if( boolean_expression1 )
        statement1;
    else if( boolean_expression2 )
        statement2;
    else
        statement3
```

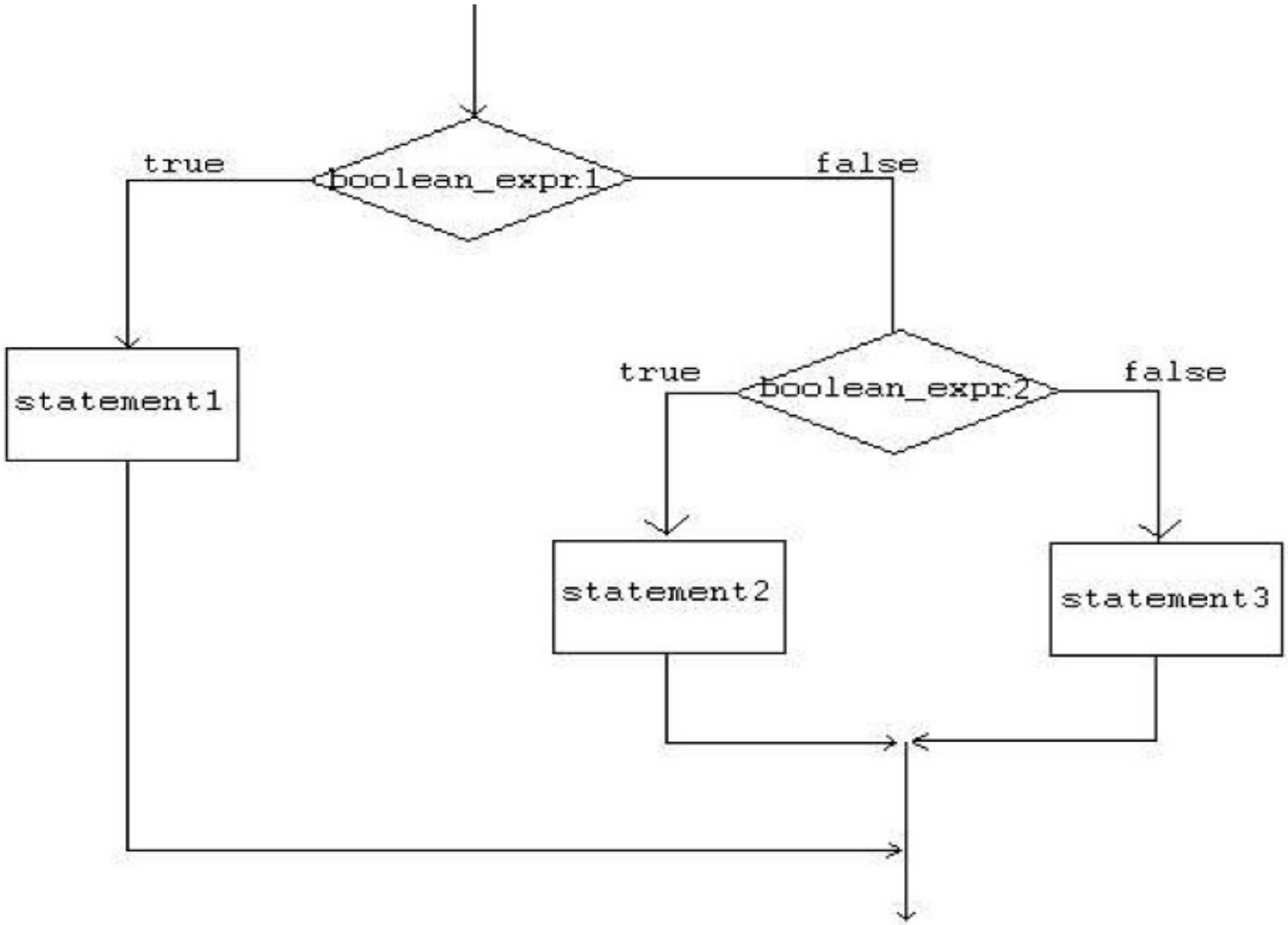
Declaración `If` Diagrama



Declaración `If else` Diagrama



Declaración `If else if` Diagrama



Errores comunes

1. La condición dentro de la declaración If no evalúa un valor boleano. Por ejemplo,

```
//WRONG
int number = 0;
if( number ){
     //some statements here
}
```

   La variable number no es un tipo booleano

2. Escribir elseif en vez de else if.

Declaración `switch`

- switch
  - La sentencia switch se encarga de estructurar una selección múltiple. Al contrario del enunciado if-else que sólo podemos indicar dos alternativas, maneja un número finito de posibilidades.
- La estructura general del enunciado switch es la siguiente:

```
switch( expresión ) {
    case constante1:
      sentencia1;
      ...
      break;
      ...
    case constanteN:
      sentenciaN;
      ...
      break;
    default:
      sentencia;
      ...
      break
}
```

Declaración `switch`

- El valor de la expresión y de las constantes tiene que ser de tipo char, byte, short o int .
- Al evaluar la expresión de switch, el intérprete busca una constante con el mismo valor.
- Si la encuentra, ejecuta las sentencias asociadas a esta constante hasta que tropiece con un break.
- La sentencia break finaliza la ejecución de esta estructura.
- Si no encuentra ninguna constante que coincida con la expresión, busca la línea default.
- Si existe, ejecuta las sentencias que le siguen. La sentencia default es opcional.

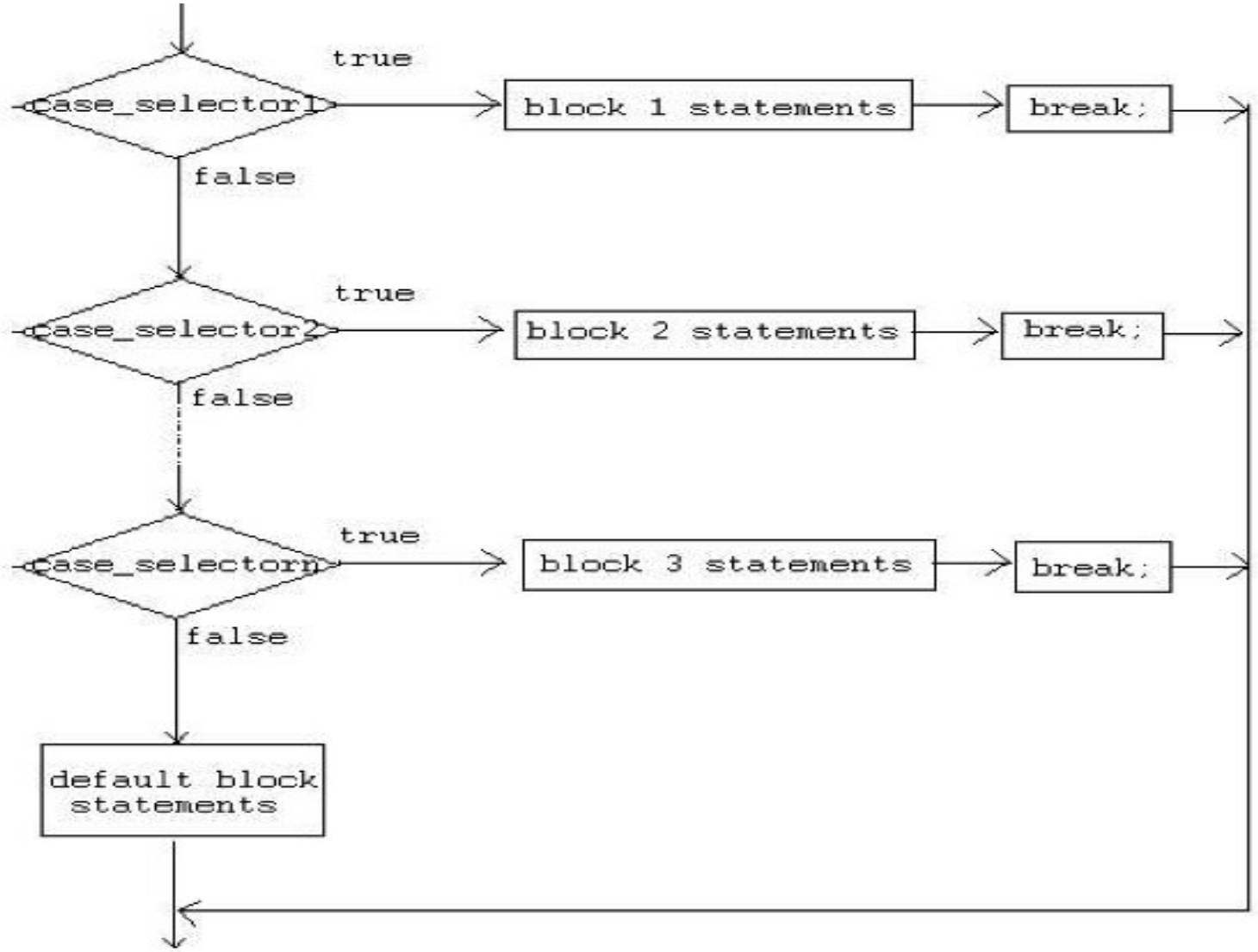
Declaración `switch`

- NOTE:
  - A diferencia con el caso de declaración if, las múltiples declaraciones se ejecutan en la declaración sin necesidad de cambiar las llaves.
  - Cuando un case en un switch se ha encontrado la misma condición, todas las declaraciones relacionadas con este case se ejecutan. No sólo eso, las declaraciones relacionadas con los cases siguientes son también ejecutados.
  - Para evitar que el programa siga ejecutando los case posteriores, usamos la declaración **break** como última declaración.

Ejemplo declaración `switch`

```
public class Lineas{
    public static void main(String args[]){
      int j = 0;
      switch (j) {
      case 5:
        System.out.println("5********");
      case 4:
        System.out.println("4********");
        break;
      case 3:
        System.out.println("3********");
      case 2:
        System.out.println("2********");
      case 1:
        System.out.println("1********");
      default:
        System.out.println(“Por defecto");
      }
    }
}
```

Diagrama de Flujo `switch`



Estructuras de iteración

- Estructuras de iteración
  - Nos permite ejecutar secciones específicas del código una cantidad determinada de veces..

- Tipos:
  - while
  - do-while
  - for

bucle while

- while
  - La iteración continuará hasta que su condición sea falsa.

- while tiene la siguiente forma:

```
while( boolean_expression ){
     statement1;
     statement2;
     . . .
}
```

Ejemplo 1

```
int x = 0;

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

Ejemplo 2

```
//infinite loop
while(true)
     System.out.println(“hello”);
```

Ejemplo 3

```
//no loops
// statement is not even executed
while (false)
     System.out.println(“hello”);
```

bucle do-while

- do-while
  - La sentencia de iteración **do-while** es de tipo posprueba.
  - Primero realiza las acciones luego pregunta.
- do-while tiene esta sintaxis:

```
do{
        statement1;
        statement2;
        . . .
}while( boolean_expression );
```

Ejemplo 1

```
int x = 0;

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

Ejemplo 2

```
//infinite loop
do{
     System.out.println(“hello”);
} while (true);
```

Ejemplo 3

```
//one loop
// statement is executed once
do
     System.out.println(“hello”);
while (false);
```

Directrices de codificación

1. Un error de programación común se da cuando se utiliza el bucle do-while y se olvidar de escribir el punto y coma después de la expresión while,

```
do{
  ...
 }while(boolean_expression)//WRONG->forgot semicolon;
```

2. Al igual que en el while, controlar entrar en un bucle infinito.

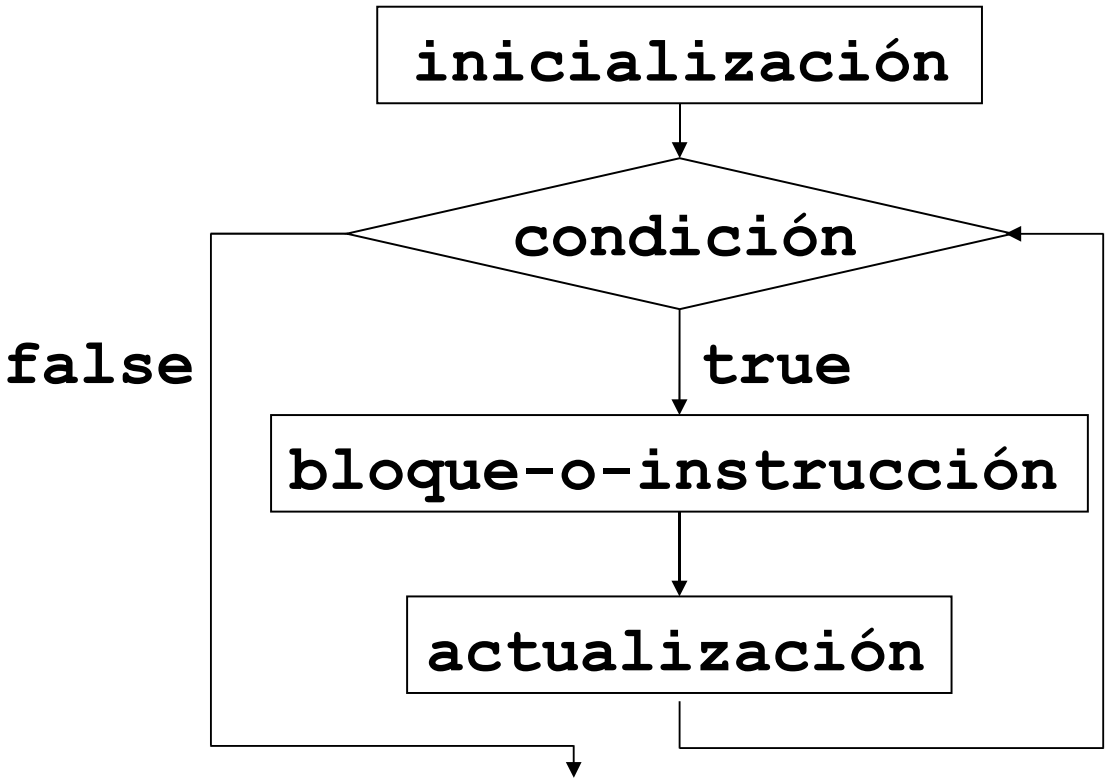
bucle `for`

- Bucle for
  - Permite la ejecución del mismo código un número de veces.
- Sintaxis del bucle for:

```
for (<inicialización>;<condición>;<actualización>)
  <bloque-o-instrucción>
```

- Siempre equivalente a un bucle **`while`**
- Como en el caso de **`do-while`**, muchas veces un bucle **`for`** es más compacto que un **`while`**.

Diagrama bucle `for`



Sentencia break

- La sentencia de 'break' es de tipo de control de bucles.

- Dentro de la iteracion en un bucle, de cualquiera de los tipos (while, do-while, for), el uso de esta sentencia rompe la iteracion de dicho bucle.
- Cuando se encuentra en una sentencia switch o bucle, break hace que el control del flujo del programa pase a la siguiente sentencia que se encuentre fuera del entorno del switch o bucle.

Sentencia break

- Ejemplo:

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

Este 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.

Sentencia break etiquetada

- Java incorpora la posibilidad de etiquetar la sentencia break, de forma que el control pasa a sentencias que no se encuentran inmediatamente después de la sentencia switch o del bucle, es decir, saltará a la sentencia en donde se encuentre situada la etiqueta.
- La sintaxis de una sentencia etiquetada es la siguiente:

```
break [etiqueta];
```

Ejemplo break etiquetada

```
public class TestBreakLabel
{
   public static void main(String[] args)
   {
      int[][] numbers = {{1, 2, 3},     {4, 5, 6,7, 8, 9}};
      int searchNum = 8;
      boolean foundNum = false;
      searchLabel:
      for( int i=0; i<numbers.length; i++ ){
            for( int j=0; j<numbers[i].length; j++ ){
                  if( searchNum == numbers[i][j] ){
                        foundNum = true;
                        break searchLabel;
                  }
            }
      }
      if( foundNum )
            System.out.println(searchNum + " found!" );
      else
            System.out.println(searchNum + " not found!");

   }
}
```

Sentencia continue

- La sentencia de **continue** es de tipo de control de bucles.
- Dentro de la iteracion en un bucle, de cualquiera de los tipos (while, do-while, for), el uso de esta sentencia rompe la iteracion de dicho bucle. La sentencia continue no se puede utilizar en una sentencia switch.
- Provocando que se ejecute la siguiente iteracion de dicho bucle, ignorando las sentencias posteriores a "continue"

Sentencia continue

- Ejemplo:

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

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 condicion del for y en caso de cumplirse de nuevo a la sentencia 1 y así sucesivamente.

Sentencia continue etiquetada

- Java permite el uso de etiquetas en la sentencia continue, de forma que el funcionamiento normal se ve alterado y el salto en la ejecución del flujo del programa se realizará a la sentencia en la que se encuentra colocada la etiqueta.
- La sintaxis de una sentencia etiquetada es la siguiente:

```
continue [etiqueta];
```

Ejemplo continue etiquetada

```
outerLoop:
for( int i=0; i<5; i++ ){
   for( int j=0; j<5; j++ ){
     System.out.println("Inside for(j) loop j="+j+"\n i="+i);//msg1
     if( j == 2 )    continue outerLoop;
   }
   System.out.println("Inside for(i) loop"); //message2
}
```

In this example, message 2 never gets printed since we have the statement continue outerloop which skips the iteration.

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. En caso de no hacerlo se generará un *Warning*, y el código Java no se puede compilar con Warnings.

Sentencia return

- 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.