Declaración Java If-else

La *instrucción* Java *if* se usa para probar la condición. Comprueba la condición booleana: *verdadero* o *falso* . Hay varios tipos de sentencias if en java.

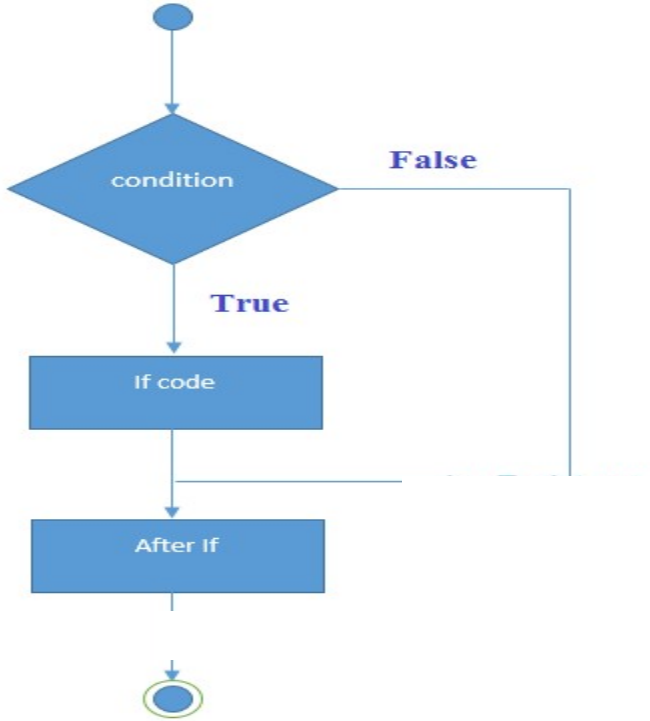
- Declaración if
- declaración if-else
- escalera if-else-if
- anidada si la declaración

Declaración Java if

La instrucción Java if prueba la condición. Ejecuta el *bloque* if si la condición es verdadera.

Sintaxis:

```
if (condición) {
// código a ejecutar
}
```

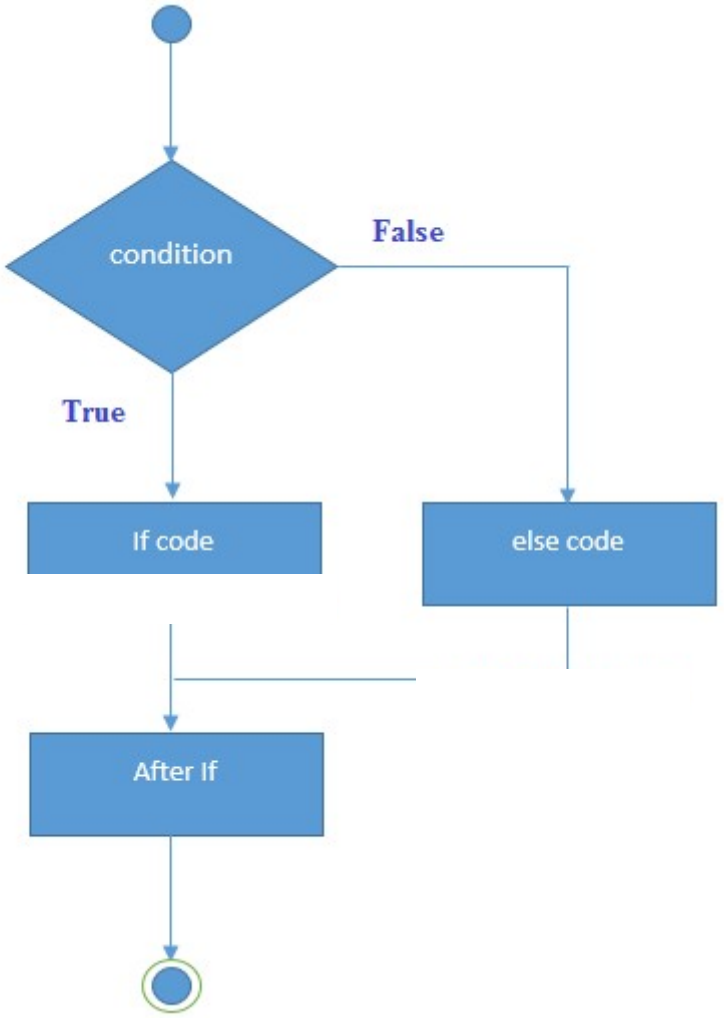


Declaración if-else de Java

La instrucción if-else de Java también prueba la condición. Ejecuta el *bloque* if si la condición es verdadera; de lo *contrario,* se ejecuta el *bloque* .

Sintaxis:

```
if (condición) {
// código si la condición es verdadera
} más {
// código si la condición es falsa
}
```



Usando operador ternario

También podemos usar el operador ternario (? :) para realizar la tarea de la declaración if ... else. Es una forma abreviada de verificar la condición. Si la condición es verdadera, ¿el resultado de? es regresado. Pero, si la condición es falsa, se devuelve el resultado de:

Sintaxis:

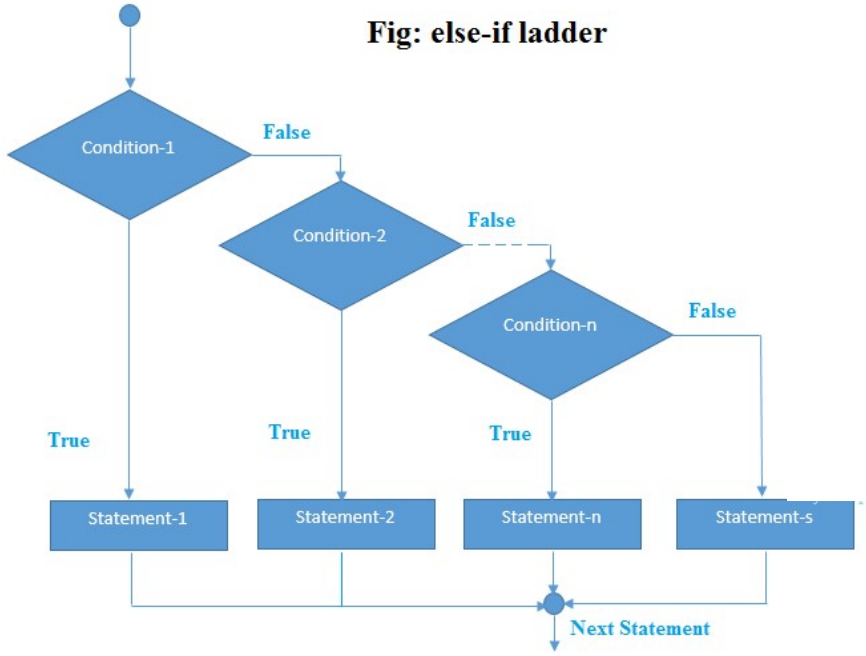
```
String cadena = (número% 2 == 0 )? "número par" : "número impar" ;
```

Declaración de escalera Java if-else-if

La instrucción de escalera if-else-if ejecuta una condición a partir de varias instrucciones.

Sintaxis:

```
1. if (condición1) {
2. // código que se ejecutará si condición1 es verdadera
3. } else if (condición2) {
4. // código que se ejecutará si la condición2 es verdadera
5. }
6. otra cosa si (Condition3) {
7. // código que se ejecutará si condition3 es verdadero
8. }
9. ...
10. más {
11. // código a ejecutar si todas las condiciones son falsas
12. }
```



Ejemplo:

Salida:

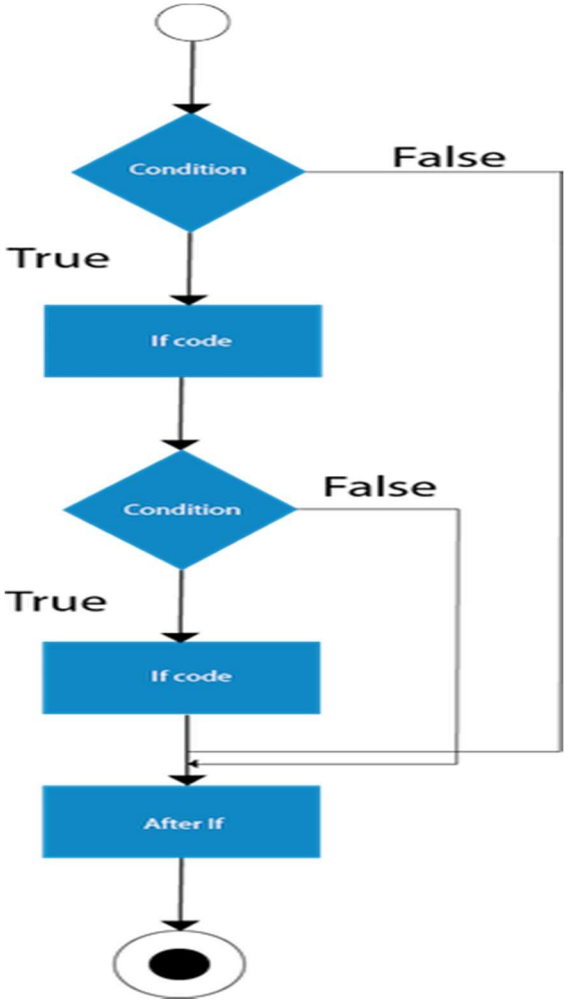
Programa para verificar POSITIVO, NEGATIVO o CERO:

Declaración anidada Java if

La instrucción if anidada representa el *bloque if dentro de otro bloque if* . Aquí, la condición de bloque if interno se ejecuta solo cuando la condición de bloque if externa es verdadera.

Sintaxis:

```
if (condición) {
    // código a ejecutar
        if (condición) {
          // código a ejecutar
    }
}
```



Ejemplo1:

Ejemplo 2

Switch

Switch ejecuta una instrucción de múltiples condiciones. Es como la declaración de escalera if-else-if. La instrucción Switch funciona con los tipos byte, short, int, long, enum, String y algunos tipos de contenedor como Byte, Short, Int y Long. Desde Java 7, puede usar cadenas en la instrucción Switch.

En otras palabras, la instrucción Switch prueba la igualdad de una variable con múltiples valores.

Puntos para recordar

- Puede haber *uno o N número de valores de caso* para una expresión de cambio.
- El valor del caso debe ser solo del tipo de expresión de cambio. El valor del caso debe ser *literal o constante* . No permite variables.
- Los valores del caso deben ser *únicos* . En caso de valor duplicado, genera un error en tiempo de compilación.
- La expresión del Switch Java debe ser *byte, short, int, long (con su tipo Wrapper), enumeraciones y cadenas* .
- Cada declaración de caso puede tener una *declaración de interrupción* que es opcional. Cuando el control llega a la declaración de interrupción, salta el control después de la expresión de cambio. Si no se encuentra una declaración de interrupción, se ejecuta el siguiente caso.
- El valor del caso puede tener una *etiqueta predeterminada* que es opcional.

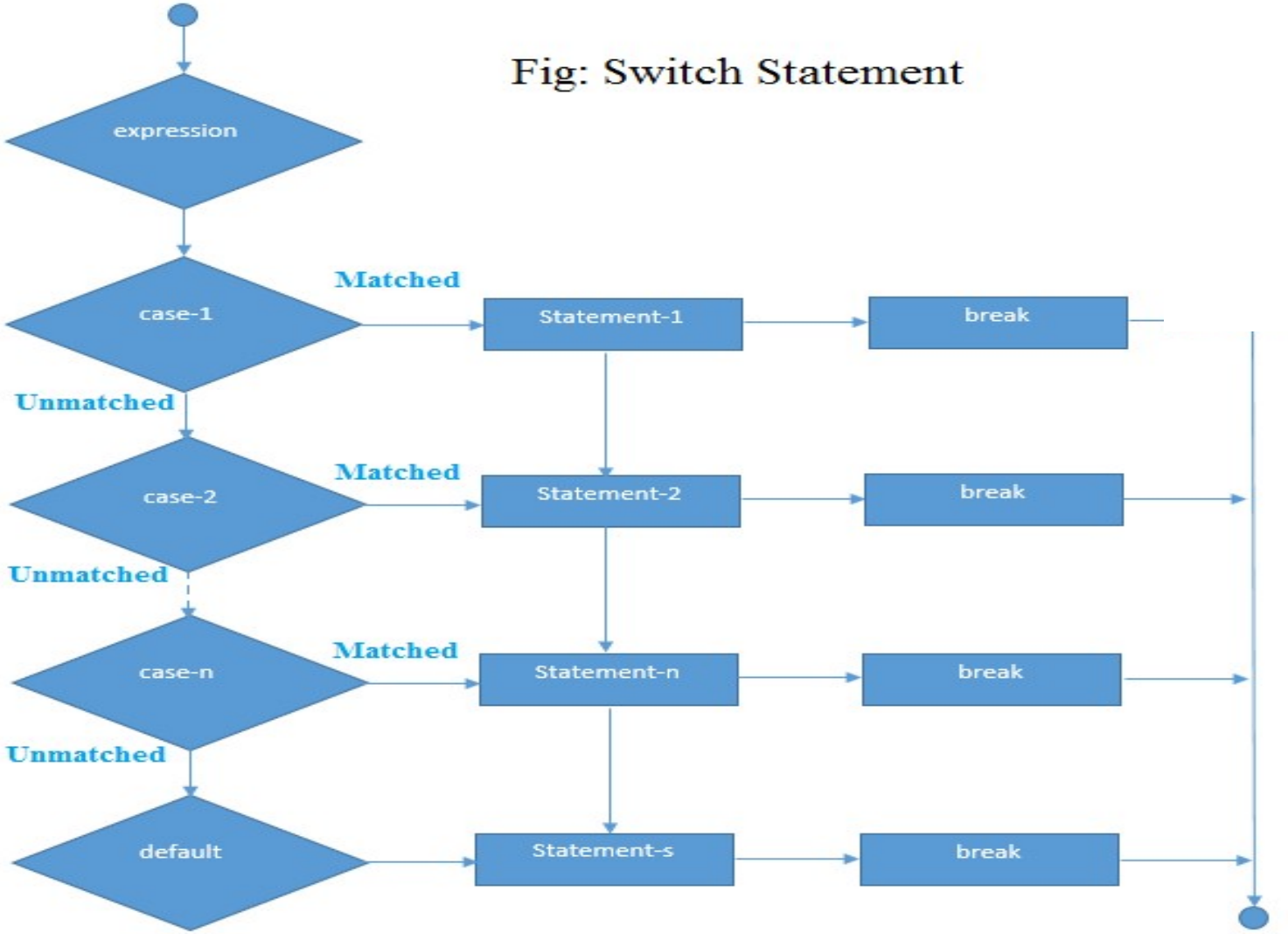
Sintaxis:

```
1. interruptor (expresión) {
2. valor del  caso1:
3.  // código a ejecutar;
4.  romper ;  //Opcional
5. valor del  caso2:
6.  // código a ejecutar;
7.  romper ;  //Opcional
8. ......
9. por defecto :
10. código que se ejecutará  si  todos los casos no coinciden; }
```

Ejemplo:

Ejemplo de mes de búsqueda:

Programa para verificar Vocal o Consonante:



La declaración Switch fallida

Significa que ejecuta todas las declaraciones después de la primera coincidencia si no hay una declaración de interrupción.

Ejemplo:

```
No en 10, 20 o 30
```

Switch con cadena

Java nos permite usar cadenas en la expresión de cambio desde Java SE 7. La declaración del caso debe ser literal de cadena.

Ejemplo:

Declaración de Switch anidado de Java

Podemos usar la instrucción Switch dentro de otra instrucción Switch en Java. Se conoce como declaración de interruptor anidado.

Ejemplo:

Java Enum en la instrucción Switch

Java nos permite usar enum en la instrucción Switch.

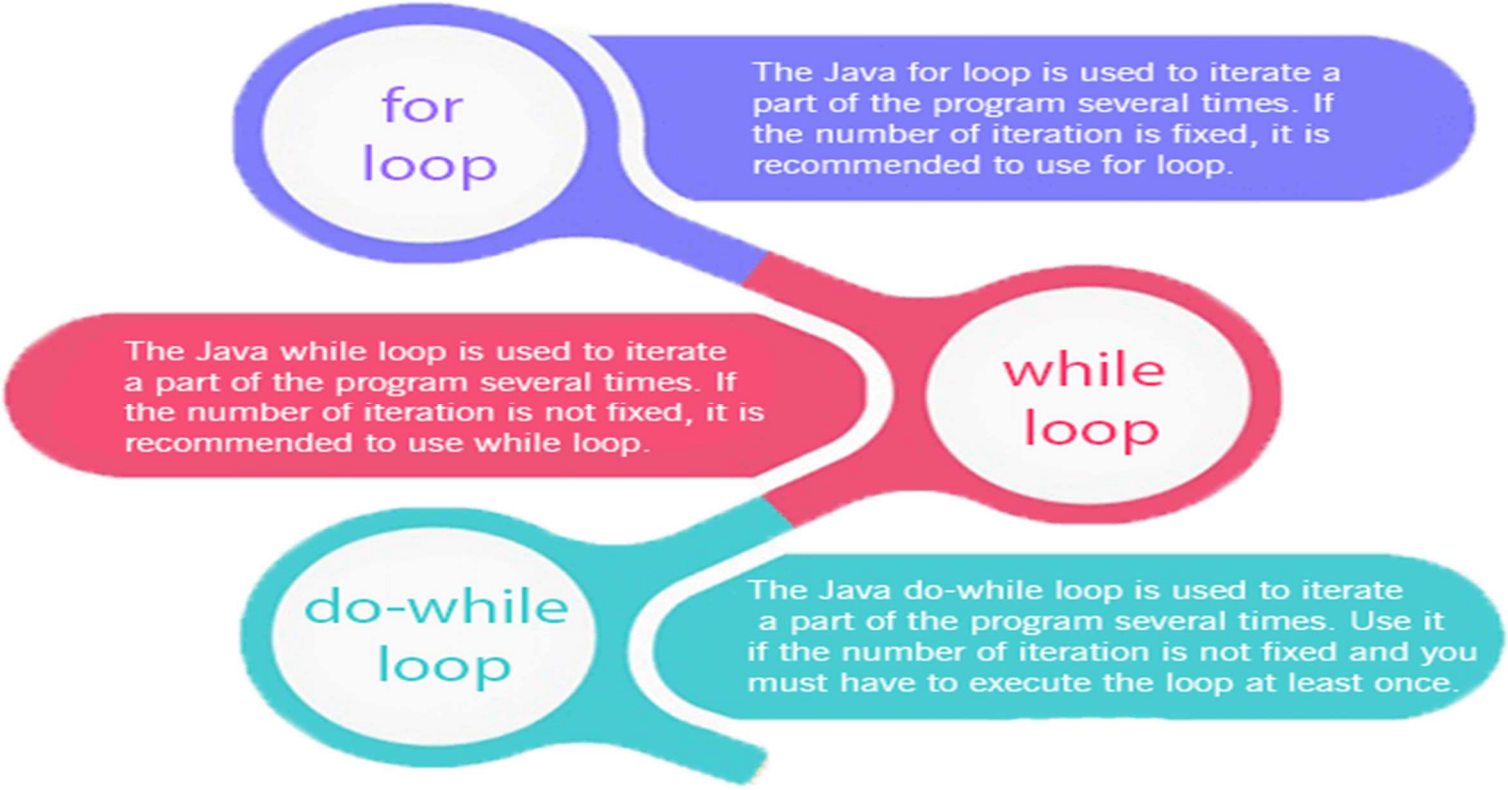
Ejemplo:

Java Wrapper en Switch

Bucles en Java

En los lenguajes de programación, los bucles se utilizan para ejecutar un conjunto de instrucciones / funciones repetidamente cuando se cumplen ciertas condiciones. Hay tres tipos de bucles en java.

- en bucle
- mientras bucle
- bucle do-while



Java For Loop vs While Loop vs Do While Loop

Comparación	en bucle	mientras bucle	hacer mientras bucle
Introducción	Java for loop es una declaración de flujo de control que itera una parte de los programas varias veces.	Es una declaración de flujo de control que ejecuta una parte de los programas repetidamente en función de una condición booleana dada.	El bucle do while de Java es una declaración de flujo de control que ejecuta una parte de los programas al menos una vez y la ejecución posterior depende de la condición booleana dada.
Cuándo usar	Si el número de iteraciones es fijo, se recomienda usar para el bucle.	Si el número de iteraciones no es fijo, se recomienda usar el ciclo while.	Si el número de iteraciones no es fijo y debe ejecutar el ciclo al menos una vez, se recomienda usar el ciclo do-while.
Sintaxis	`para (init; condición; incr / decr) { // código a ejecutar }`	`while (condición) { // código a ejecutar }`	`hacer{ // código a ejecutar } while (condición);`
Ejemplo	`//en bucle para (int i = 1; i <= 10; i ++) { System.out.println (i); }`	`// while loop int i = 1; mientras que (i <= 10) { System.out.println (i); i ++; }`	`// ciclo do-while int i = 1; hacer{ System.out.println (i); i ++; } while (i <= 10);`
Sintaxis para bucle infinitivo	`para(;;){ // código a ejecutar }`	`while (verdadero) { // código a ejecutar }`	`hacer{ // código a ejecutar } while (verdadero);`

Java For Loop

Se usa para iterar una parte del programa varias veces. Si el número de iteraciones es fijo, se recomienda usar para el bucle.

Hay tres tipos de bucles for en java.

- Simple For Loop
- For-each o Enhanced For Loop
- Etiquetado para Loop

Java simple para bucle

Un bucle for simple es lo mismo que C / C ++. Podemos inicializar la variable, verificar la condición y el valor de incremento / decremento. Consta de cuatro partes:

1. **Inicialización** : es la condición inicial que se ejecuta una vez cuando se inicia el ciclo. Aquí, podemos inicializar la variable, o podemos usar una variable ya inicializada. Es una condición opcional.
2. **Condición** : es la segunda condición que se ejecuta cada vez para probar la condición del bucle. Continúa la ejecución hasta que la condición sea falsa. Debe devolver un valor booleano verdadero o falso. Es una condición opcional.
3. **Declaración** : la declaración del bucle se ejecuta cada vez hasta que la segunda condición sea falsa.
4. **Incremento / Decremento** : Incrementa o disminuye el valor de la variable. Es una condición opcional.

Sintaxis: `for` (inicialización; condición; incr / decr) { // sentencia o código a ejecutar }

Java anidado para bucle

Si tenemos un bucle for dentro del otro bucle, se conoce como bucle for anidado. El bucle interno se ejecuta completamente cada vez que se ejecuta el bucle externo.

Ejemplo:

Pirámide Ejemplo 1:

Pirámide Ejemplo 2:

Java para cada bucle

El ciclo for-each se usa para atravesar una matriz o colección en java. Es más fácil de usar que simple para el bucle porque no necesitamos incrementar el valor y usar la notación de subíndice.

Funciona sobre la base de elementos no indexados Devuelve el elemento uno por uno en la variable definida.

Sintaxis:

Java labeled for

Podemos tener un nombre de cada Java para el bucle. Para hacerlo, usamos la etiqueta antes del ciclo for. Es útil si hemos anidado el bucle para que podamos romper / continuar específicamente para el bucle.

Sintaxis:

1. Nombre de etiqueta:
2. **para** (inicialización; condición; incr / decr) { // código a ejecutar }

Ejemplo:

Java bucle infito for

Si usa dos puntos y coma ;; en el ciclo for, será infinitivo para el ciclo.

Sintaxis:

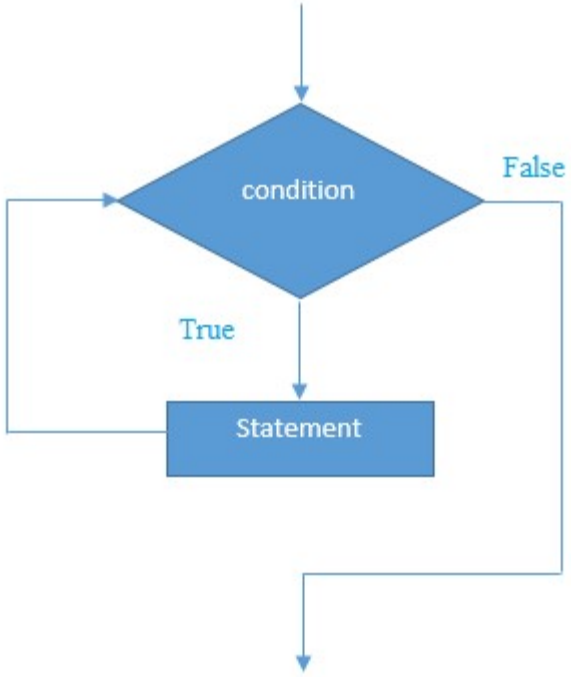
```
para (;;) {
// código a ejecutar
}
```

Ejemplo:

Ahora, debe presionar ctrl + c para salir del programa.

Java While Loop

El *bucle while de* Java se usa para iterar una parte del programa varias veces. Si el número de iteraciones no es fijo, se recomienda usar el ciclo while.



Sintaxis:

```
while (condición) {
// código a ejecutar
}
```

Ejemplo:

Java Infinitive While Loop

Si pasa **true** en el ciclo while, será infinitivo mientras que el ciclo.

Sintaxis:

```
1. while ( verdadero ) {
2. // código a ejecutar
3. }
```

Ejemplo:

Salida:

Java do-while Loop

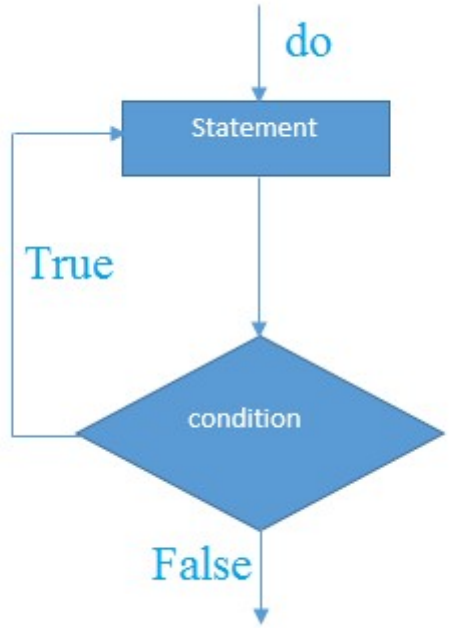
El *bucle do-while de* Java se usa para iterar una parte del programa varias veces. Si el número de iteraciones no es fijo y debe ejecutar el bucle al menos una vez, se recomienda usar el bucle do-while.

El *bucle do-while de* Java se ejecuta al menos una vez porque la condición se verifica después del cuerpo del bucle.

Sintaxis:

```
1. hacer {
2. // código a ejecutar
3. } while (condición);
```

Ejemplo:



Java Infinitive Do-while Loop

Si pasa **true** en el ciclo do-while, será infinito el ciclo do-while.

Sintaxis:

```
1. hacer {
2. // código a ejecutar
3. } while ( verdadero );
```

Ejemplo:

Break

Cuando se encuentra una declaración de interrupción dentro de un bucle, el bucle finaliza inmediatamente y el control del programa se reanuda en la siguiente instrucción que sigue al bucle.

El Java *ruptura* se utiliza para romper bucle o sentencia Switch. Rompe el flujo actual del programa en la condición especificada. En caso de bucle interno, solo rompe el bucle interno.

Podemos usar la declaración de interrupción de Java en todos los tipos de bucles, como for loop, while loop y do-while loop.

Sintaxis:

1. declaración de salto;
2. **break** ;

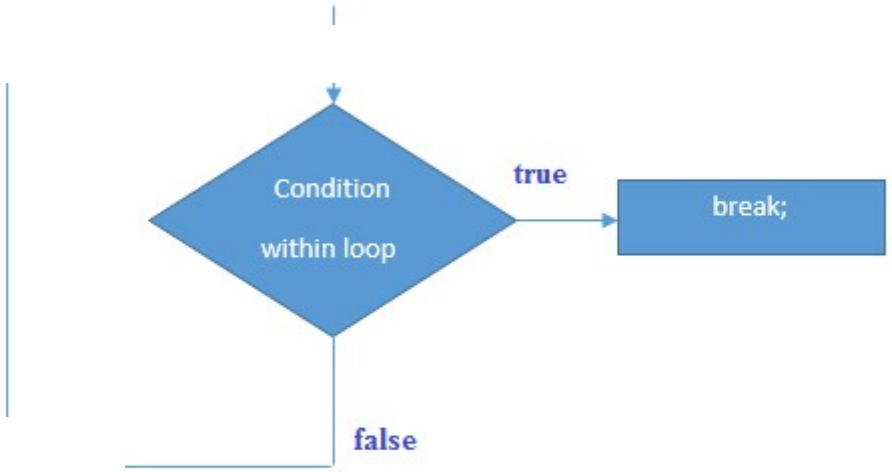


Figure: Flowchart of break statement