INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

JAVA 8

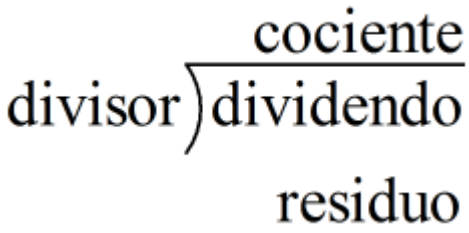
APACHE NETBEANS IDE 12.5

Condicionales y Bucles

Contenido

Operadores Matemáticos

- **Módulo:** Este operador matemático lo que hace es devolverme el residuo (también llamado módulo) de una división:



- **incremento:** Sirve para sumar o restar cierto número, para aumentar o reducir el valor de una variable.

  Con el operador **++ aumenta uno**, con el operador **+= n aumenta n lugares**.

  Con el operador **-- reduce uno**, con el operador **-= n se reduce n lugares**.

OPERADOR	NOMBRE
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo
++ +=	Incremento
-- -=	Decremento

Operadores Lógicos de Comparación

Sirven para comparar el valor de dos variables o constantes para devolver finalmente como resultado un valor booleano de **true** o **false**.

- **== Igual que:** Esta operación lógica retorna un valor booleano de verdadero (**true**) si las dos variables son iguales, aunque sean de diferente tipo de dato.

- **=== Idéntico que:** Esta operación lógica retorna un valor booleano de verdadero (**true**) si las dos variables son iguales y del mismo tipo de dato.
- **!= Diferente que:** Esta operación lógica retorna un valor booleano de verdadero (**true**) si las dos variables NO son iguales.
- **<> Diferente que:** Esta operación lógica retorna un valor booleano de verdadero (**true**) si las dos variables NO son iguales y además no son del mismo tipo de dato.
- **< Menor que:** Esta operación lógica retorna un valor booleano de verdadero (**true**) si el valor de la variable del lado izquierdo es menor a la del lado derecho del símbolo.
- **> Mayor que:** Esta operación lógica retorna un valor booleano de verdadero (**true**) si el valor de la variable del lado izquierdo es mayor a la del lado derecho del símbolo.
- **<= Menor o igual que:** Esta operación lógica retorna un valor booleano de verdadero (**true**) si el valor de la variable del lado izquierdo es menor o igual a la del lado derecho del símbolo.
- **>= Mayor o igual que:** Esta operación lógica retorna un valor booleano de verdadero (**true**) el valor de la variable del lado izquierdo es mayor o igual a la del lado derecho del símbolo.

Operadores Lógicos

Sirven para comparar el resultado booleano de dos o más operaciones lógicas de comparación, para devolver finalmente como resultado un valor booleano de **true** o **false**.

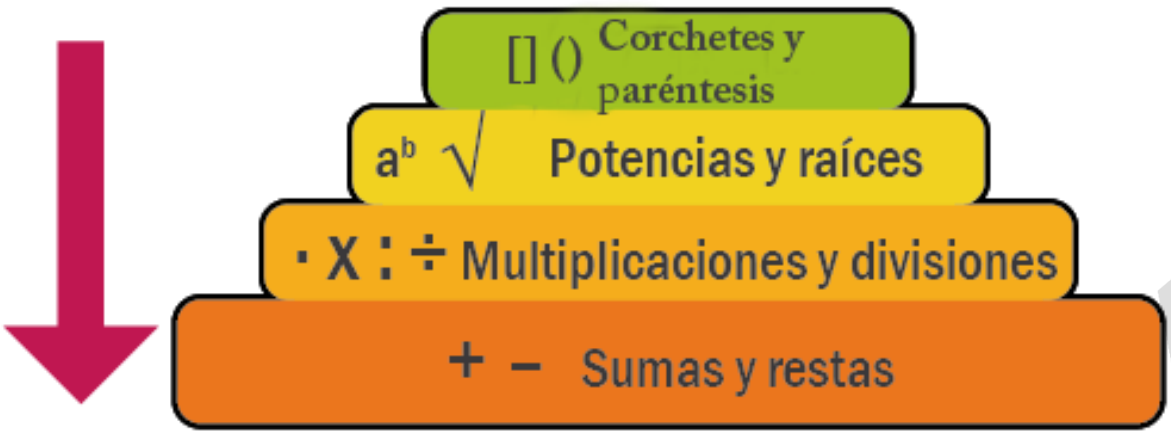
- **&&, Y lógico:** Establece que el resultado es **true** **solamente cuando ambos operadores de comparación se cumplen** (osea si ambos son **true**), tiene **mayor jerarquía** o prioridad **que el operador and**.
- **and, Y lógico:** Establece que el resultado es **true** **solamente cuando ambos operadores de comparación se cumplen**, tiene **menor jerarquía** o prioridad **que el operador &&.**
- **||, Ó lógico:** Establece que el resultado es **true** **cuando alguno de los operadores de comparación se cumple o cuando ambos se cumplen**, tiene **mayor jerarquía** o prioridad **que el operador OR**.
- **or, Ó lógico:** Establece que el resultado es **true** **cuando alguno de los operadores de comparación se cumple**, tiene **menor jerarquía** o prioridad **que el operador ||**.
- **xor, Ó exclusivo:** El resultado es **true** **solamente cuando uno de los operadores de comparación se cumple** (osea si solo uno es **true**), **si ambos operadores de comparación se cumplen da como resultado false**.

- **!, Negación:** Hace que el resultado a la derecha del símbolo de exclamación sea lo opuesto a lo que es, por lo tanto, si el resultado de la operación con los operadores de comparación era **true**, convierte el resultado a **false** y viceversa.
- **=, Asignación:** Le da el valor de lo que sea que se encuentre a su derecha a cualquier variable.

OPERADOR	NOMBRE
&&	Y LÓGICO
AND	Y LÓGICO
||	O LÓGICO
OR	O LÓGICO
XOR	O EXCLUSIVO
!	NEGACIÓN (NOT)

Jerarquía de Operaciones Matemáticas

A la hora de utilizar operadores aritméticos se debe de tener en cuenta la prioridad o jerarquía de operaciones, esto significa que el programa les hará caso a algunas operaciones primero que a otras, la jerarquía de operaciones matemáticas es la siguiente:



- EJEMPLO:
  - 2 + 3 x 6 = 20
  - (2 + 3) x 6 = 30

Condicionales

Los condicionales son estructuras que analizan una operación lógica para ejecutar cierto código o no dependiendo del resultado de dicha operación, si no se cumple el resultado fijado para que se ejecute el condicional, el programa simplemente ignora esas líneas de código como si no existieran. Existen 3 tipos de condicionales en Java:

1. Condicional **if.**
2. Condicional **else if.**
3. Condicional **switch-case**.

Condicional if

- Este condicional lo que hace es analizar **una sola condición**, evaluando lo que tiene dentro de su paréntesis y efectuando el código que tiene en su interior solo si el resultado o valor de la variable booleana es **true**, si el resultado es **false** ejecuta lo que haya en las llaves del **else** y si no existe **else** el programa simplemente ignora al condicional y se lo brinca para seguir con el flujo de ejecución normal.

```
if(Condición a evaluar utilizando operadores lógicos, con valor final true o false){
    Código por ejecutar si lo que hay dentro del paréntesis del if es true
}else{
    Código por ejecutar si lo que hay dentro del paréntesis del if es false
    }
```

Condicional else if

- Este condicional lo que hace es analizar **más de una sola condición que esté relacionada con otra**, evaluando lo que tenga dentro de su paréntesis y efectuando el código que esté en su interior si el resultado de la operación lógica es **true**, pero a diferencia del **if** normal, cuando la primera condición no se cumpla, el programa procederá a evaluar la condición del primer **else if** y si no se cumple tampoco, se irá a analizar la condición del siguiente **else if**, así hasta que ya no queden condicionales para evaluar. Si el resultado de todas las operaciones lógicas dentro del paréntesis del condicional **if** y todos los **else if** es **false**, el programa ejecuta lo que haya en las llaves del **else** y si no existe **else** el programa simplemente ignora al condicional y se lo brinca para seguir con el flujo de ejecución normal.

if(**Condición 1**){

Código por ejecutar si lo que hay dentro del paréntesis del if es **true**

}**else if**(**Condición 2 relacionada a la 1**){

Código por ejecutar cuando lo que haya dentro del paréntesis del **else if** sea **true** y lo que haya dentro del paréntesis del **if** sea **false**

}**else if**(**Condición 3 relacionada a la 2**){

Código por ejecutar cuando lo que haya dentro del paréntesis del segundo **else if** sea **true** y lo que haya dentro del paréntesis del **if** y del primer **else if** sea **false**

}**else**{

Código por ejecutar cuando lo que haya dentro del paréntesis de todo lo anterior sea **false**

}

Sirve por ejemplo para evaluar una edad cualquiera y determinar si ésta es menor a 18, se encuentra entre 18 y 40, entre 40 y 60 años o es mayor a 60. En este ejemplo podemos ver que la edad la determinaremos dependiendo de varias condiciones que están relacionadas entre sí.

x <18< x <40< x <60< x

Para cada rango en el que se encuentre la edad se ejecutará cierta acción:

if(edad<18) {

Código por ejecutar si la edad es menor a 18 años, osea que la condición **x <18** del if sea **true**

}**else if**(edad<40){

Código por ejecutar si la edad es mayor a 18 años y menor a 40 años, osea que la condición

18< x <40 del else if sea **true** y que la condición del if sea **false**

}**else if**(edad<60){

Código por ejecutar si la edad es mayor a 40 años y menor a 60 años, osea que la condición

40< x <60 del else if sea **true** y que la condición del if y del primer else if sea **false**

}**else**{

Código por ejecutar si las condiciones de todo lo anterior es **false** por lo que la condición que se cumple será que **60 < x**

}

Condicional Switch-Case

- Este condicional primero que nada analiza la operación lógica que se encuentra dentro de su paréntesis, si el resultado de esa operación es un valor booleano **true**, **analiza el valor de la variable** del paréntesis del **switch** **para compararlo con una serie de valores pre-programados y enlistados que puede adoptar, si su valor coincide con alguno de los existentes de los case, se ejecutará el código correspondiente, sino se ejecutará la línea de código de la instrucción Default** ,cuando el valor de la variable coincida con el valor de un **case** se ejecutará ese código y al llegar a la instrucción de **Break**, el programa se saldrá del condicional y seguirá con el flujo de ejecución fuera del condicional. Si el resultado de la operación lógica en un inicio es **false**, el programa ignora el código completamente y se salta a la siguiente línea de código fuera del condicional.

```
switch (Variable a evaluar) {

    Case valor 1:

    Código por ejecutar cuando la variable a evaluar del condicional adopte el valor 1

    Break;


    Case valor 2:

    Código por ejecutar cuando la variable a evaluar del condicional adopte el valor 2

    Break; ...


    Case valor n:

    Código por ejecutar cuando la variable a evaluar del condicional adopte el valor n

    Break;


    Default: Código a ejecutar para cuando la variable adopte cualquier otro valor

}
```

La instrucción Deafult al igual que else puede existir o no y el código funciona correctamente aún así.

- Si para utilizar este condicional se cuenta con una forma para que el usuario ingrese datos al programa, **se puede usar para poder elegir y ejecutar alguna acción pre-programada del código en forma de menú**, para ello primero se evalúa lo que tenga dentro del paréntesis el condicional **switch** para luego ejecutar el código que tenga en su interior si el resultado de la operación lógica es **true,** luego el usuario podrá seleccionar qué opción de código quiere que sea ejecutada por medio de los **case**, si el resultado de la operación lógica es **false** el programa ignora el código completamente y se salta a la siguiente línea de código fuera del condicional.

En cada case se ejecutará el código y al llegar a la instrucción de **Break**, el programa se saldrá del switch y seguirá con el flujo de ejecución fuera del condicional.

```
switch (operación lógica a evaluar, con valor final true o false) {
    Case valor 1:
    Código por ejecutar si la edad es menor a 18 años, osea que la condición x <18 del if sea true
    Break;

    Case valor 2:
    Código por ejecutar cuando el usuario seleccione el valor 2
    Break; ...

    Case valor n:
    Código por ejecutar cuando el usuario seleccione el valor n
    Break;
    Default: Código a ejecutar para cuando el usuario seleccione cualquier otro valor;
}
```

En sí la mayor diferencia entre el condicional **switch-case** y el **else if** es que el **switch-case** es mejor usarlo cuando los valores que pueda tomar la variable a evaluar sean finitos y/o conocidos. El condicional **else if** puede declarar varias condiciones a evaluar y no debo saber exactamente qué valores puede tomar la variable a evaluar.

Si quisiéramos adecuar la función del **else if** que habíamos hecho para los números para hacerla con un **switch-case**, lo que deberíamos hacer es lo siguiente:

```
if(edad<18) {
    Código por ejecutar si la edad es menor a 18 años, osea que la condición x <18 del if sea true
} else if(edad<40) {
    Código por ejecutar si la edad es mayor a 18 años y menor a 40 años, osea que la condición
    18< x <40 del else if sea true y que la condición del if sea false
} else if(edad<60) {
    Código por ejecutar si la edad es mayor a 40 años y menor a 60 años, osea que la condición
```

40< x <60 del else if sea **true** y que la condición del if y del primer else if sea **false**

} **else** {

Código por ejecutar si las condiciones de todo lo anterior es **false** por lo que la condición que se cumple será que **60 < x**

}

Debemos poner **true** en el paréntesis del **switch-case** para asegurarnos que siempre se vaya a correr y luego en los case poner los condicionales que habíamos puesto en los **else if**.

switch(**true**) {

Case edad<18**:**

Código por ejecutar si la edad es menor a 18 años, osea que la condición **x <18** del if sea **true**

Break;

Case edad<40**:**

Código por ejecutar si la edad es mayor a 18 años y menor a 40 años, osea que la condición

18< x <40 del else if sea **true** y que la condición del if sea **false**

Break; ...

Case edad<60**:**

Código por ejecutar si la edad es mayor a 40 años y menor a 60 años, osea que la condición

40< x <60 del else if sea **true** y que la condición del if y del primer else if sea **false**

Break; ...

Default: Código por ejecutar si las condiciones de todo lo anterior es **false** por lo que la condición que se cumple será que **60 < x**

}

Bucles

Los bucles son estructuras que analizan una operación lógica para ejecutar cierto código de manera repetitiva, esto porque algunos programas tienen la necesidad de que un mismo código se ejecute

varias veces, el código se ejecutará o no dependiendo del resultado de la operación lógica que está en su paréntesis y existen 4 tipos, divididos en dos categorías:

I. Bucles determinados.
   a) Bucle **for**.

II. Bucles indeterminados.
   a) Bucle **while**.
   b) Bucle **do while**.

Bucles Determinados

Son aquellos donde sabemos cuántas veces se ejecutará el código del Bucle.

Bucle for

- **Este bucle tiene 3 elementos dentro de su paréntesis**: **el número de iniciación del bucle**, **la condición** (que dicta el número de veces que se ejecutará el código) **y** **el incremento o decremento** (son operadores matemáticos que dictan de cuánto en cuánto aumentará el número de iniciación hasta llegar al límite dictado por la condición), **todo separado por un punto y coma**.

```
for (iniciación bucle; condición; incremento/decremento) {
    Código a ejecutar mientras el resultado de la condición del for sea true
}
```

Esto funciona porque cada vez que el bucle acaba y la condición se sigue cumpliendo, el programa llega a la llave de cierre del bucle y después regresa a la parte donde está el paréntesis para aplicarle el incremento o decremento a la variable de iniciación y repetir el ciclo hasta que la condición del bucle se deje de cumplir, osea que el resultado de su operación lógica sea **false**.

Usualmente la estructura más utilizada para que se ejecute n veces un código es la siguiente:

```
for (i=0; i<n; i++) {
    Código a ejecutar mientras el resultado de la condición del for sea true
}
```

Bucle for each

- **Este bucle está creado específicamente para manejar arrays,** tiene 2 elementos dentro de su paréntesis y son los siguientes: **una variable que sea del mismo tipo que el array que queremos recorrer** y **el nombre de la matriz que quiero recorrer**, **todo separado por dos puntos**.

```
for (tipo_primitivo   nombre_variable_del_bucle   :      nombre_array) {
    Lo que quiero que le haga a la matriz que recorra true
}
```

Usualmente la estructura más utilizada para imprimir en consola los elementos de un array que tenga almacenadas palabras (osea objetos String) es la siguiente:

```
for (String      nombre_variable    :      nombre_array) {
    Esto imprimirá en consola todos los elementos del array nombre_array de tipo String
    System.out.println(nombre_variable);
}
```

Bucles indeterminados

Son aquellos donde no sabemos cuántas veces se ejecutará el código del Bucle.

Bucle while

- Este bucle analiza **la condición que está entre sus paréntesis** para poder ejecutar el código que tenga en su interior, **mientras** el resultado de la operación lógica con los operadores de comparación o el valor de la variable booleana sea **true**, el código se estará corriendo una y otra vez hasta que el resultado de la condición sea **false**, si el resultado es **false** en un inicio, el programa simplemente ignora al bucle y se lo brinca para seguir con el flujo de ejecución normal.

while(**Condición a evaluar, con valor final true o false**){

Código por ejecutar mientras lo que haya dentro del paréntesis del while sea **true**.

Usualmente se usa una operación matemática dentro para hacer que en algún punto el programa pueda salir del bucle porque si no el bucle se ejecuta infinitamente.

}

Bucle do while

- Este bucle **funciona igual que el bucle while**, la única diferencia es que este bucle ejecuta el código de su interior al menos una vez si el resultado de la operación es **false** en un inicio, ya después de haber ejecutado el programa una vez, si la condición sigue siendo **false** el programa se brincará al bucle para seguir con el flujo de ejecución normal.

do {

Código por ejecutar mientras lo que haya dentro del paréntesis del do while sea **true**

Por ser do while el código se ejecutará al menos una vez aun cuando la condición sea **false** en un inicio, si después de esa vez el resultado sigue siendo **false**, ahora sí el programa se saldrá del bucle.

} **while** (**Condición a evaluar, con valor final true o false**);

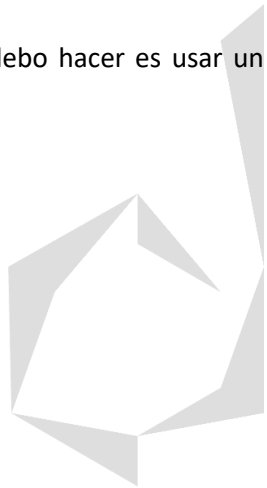
Sentencia de interrupción del Bucles

Si por alguna razón quiero interrumpir la ejecución de cualquier bucle, lo que debo hacer es usar un condicional y dentro de éste usar la instrucción **break;**

for (**iniciación bucle; condición; incremento/decremento**) {

Código a ejecutar mientras el resultado de la condición del for sea **true**

if (**Condición a evaluar, con valor final true o false**) {



```
            break;
        }
}

while (Condición a evaluar, con valor final true o false) {
        Código por ejecutar mientras lo que haya dentro del paréntesis del while sea true
        if (Condición a evaluar, con valor final true o false) {
            break;
        }
}

do {
        Código por ejecutar mientras lo que haya dentro del paréntesis del do while sea true
        if (Condición a evaluar, con valor final true o false) {
            break;
        }
} while (Condición a evaluar, con valor final true o false);
```

Referencias:

Pildoras Informáticas, “Curso Java”, 2023 [Online], Available: https://www.youtube.com/playlist?list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2lk