Semana 04

Operadores en Java

Java proporciona muchos tipos de operadores que se pueden utilizar según la necesidad. Se clasifican en función de la funcionalidad que proporcionan. Algunos de los tipos son:

1. Operadores aritméticos
2. Operadores unarios
3. Operadores de asignación
4. Operadores relacionales
5. Operadores lógicos

1. Operadores aritméticos: Se utilizan para realizar operaciones aritméticas simples en tipos de datos primitivos.

- ***:** Multiplicación
- **/:** División
- **%:** Módulo
- **+:** Adición
- **-:** Resta

2. Operadores unarios: Los operadores unarios solo necesitan un operando. Se utilizan para incrementar, decrementar o negar un valor.

- **-: Menos unario**, se usa para negar los valores.
- **+: Adición unaria**, indica el valor positivo (los números son positivos sin esto, sin embargo). Realiza una conversión automática a int cuando el tipo de su operando es byte, char o short. Esto se llama promoción numérica unaria.
- **++: Operador de incremento**, se utiliza para incrementar el valor en 1. Hay dos variedades de operadores de incremento.
    - **Post-incremento:** El valor se usa primero para calcular el resultado y luego se incrementa.
    - **Pre-incremento:** El valor se incrementa primero y luego se calcula el resultado.
- **--: Operador de decremento**, se utiliza para decrementar el valor en 1. Hay dos tipos de operadores de decremento.
    - **Post-decremento:** El valor se usa primero para calcular el resultado y luego se decrementa.
    - **Pre-decremento:** El valor se decrementa primero y luego se calcula el resultado.

- **!: Operador de NO lógico**, se usa para invertir un valor lógico.

3. Operadores de asignación: "=" El operador de asignación se utiliza para asignar un valor a cualquier variable. Tiene una asociatividad de derecha a izquierda, es decir, el valor dado en el lado derecho del operador se asigna a la variable de la izquierda y, por lo tanto, el valor del lado derecho debe declararse antes de usarlo o debe ser una constante.

El formato general del operador de asignación es:

variable = value;

En muchos casos, el operador de asignación se puede combinar con otros operadores para crear una versión más corta de la declaración llamada **Declaración compuesta**. Por ejemplo, en lugar de a = a+5, podemos escribir a += 5.

- **+=**, para sumar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.
- **-=**, para restar el operando derecho del operando izquierdo y luego asignarlo a la variable de la izquierda.
- ***=**, para multiplicar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.
- **/=**, para dividir el operando izquierdo por el operando derecho y luego asignarlo a la variable de la izquierda.
- **%=**, para asignar el módulo del operando izquierdo por el operando derecho y luego asignarlo a la variable de la izquierda.

4. Operadores relacionales: Estos operadores se utilizan para verificar relaciones como igualdad, mayor que y menor que. Devuelven resultados booleanos después de la comparación y se utilizan ampliamente en instrucciones de bucle, así como en declaraciones condicionales if-else. El formato general es,

variable **relation_operator** value

Algunos operadores de relación son:

- **==, relación de igualdad** devuelve verdadero si el lado izquierdo es igual al lado derecho.
- **!=, relación de no igualdad** devuelve verdadero si el lado izquierdo no es igual al lado derecho.
- **<, menor que** devuelve verdadero si el lado izquierdo es menor que el lado derecho.
- **<=, menor o igual que** devuelve verdadero si el lado izquierdo es menor o igual que el lado derecho.

- **>, mayor que** devuelve verdadero si el lado izquierdo es mayor que el lado derecho.
- **>=, mayor o igual que** devuelve verdadero si el lado izquierdo es mayor o igual que el lado derecho.

5. Operadores lógicos: Estos operadores se utilizan para realizar operaciones de "Y lógico" y "O lógico", es decir, una función similar a la puerta AND y la puerta OR en la electrónica digital. Una cosa a tener en cuenta es que la segunda condición no se evalúa si la primera es falsa, es decir, tiene un efecto de cortocircuito. Se usa ampliamente para probar varias condiciones para tomar una decisión. Java también tiene "NO lógico", que devuelve verdadero cuando la condición es falsa y viceversa.

Operadores lógicos:

- **&&, AND lógico:** devuelve verdadero cuando ambas condiciones son verdaderas.
- **||, OR lógico:** devuelve verdadero si al menos una condición es verdadera.
- **!, NOT lógico:** devuelve verdadero cuando una condición es falsa y viceversa.

Precedencia y Asociatividad

Las reglas de precedencia y asociativas se utilizan cuando se trata de ecuaciones híbridas que involucran más de un tipo de operador. En tales casos, estas reglas determinan qué parte de la ecuación considerar primero, ya que puede haber muchas valoraciones diferentes para la misma ecuación. La siguiente tabla muestra la precedencia de los operadores en orden decreciente como magnitud, con la parte superior representando la precedencia más alta y la inferior mostrando la precedencia más baja.

Operators	Associativity	Type
++ --	Right to left	Unary postfix
++ -- + - ! (type)	Right to left	Unary prefix
/ * %	Left to right	Multiplicative
+ -	Left to right	Additive
< <= > >=	Left to right	Relational
== !==	Left to right	Equality
&	Left to right	Boolean Logical AND
^	Left to right	Boolean Logical Exclusive OR
|	Left to right	Boolean Logical Inclusive OR
&&	Left to right	Conditional AND
||	Left to right	Conditional OR
?:	Right to left	Conditional
= += -= *= /= %=	Right to left	Assignment

Aspectos interesantes en operadores en código

1. Precedencia y asociatividad: A menudo hay confusión cuando se trata de ecuaciones híbridas que son ecuaciones que tienen múltiples operadores. El problema es qué parte resolver primero. Hay una regla de oro a seguir en estas situaciones. Si los operadores tienen precedencia diferente, resuelva primero la precedencia más alta. Si tienen la misma precedencia, resuelve según la asociatividad, es decir, de derecha a izquierda o de izquierda a derecha. La explicación del programa a continuación está bien escrita en comentarios dentro del propio programa.

```
public class Operadores {
	public static void main(String[] args)
	{
		int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;

		// Reglas de precedencia para operadores aritméticos.
		// (* = / = %) > (+ = -)
		// Imprime a+(b/d)
		System.out.println("a+b/d = " + (a + b / d));

		// Si es la misma precedencia entonces es asociativa.
		// Con las siguientes reglas.
		// e/f -> b*d -> a+(b*d) -> a+(b*d)-(e/f)
		System.out.println("a+b*d-e/f = "
```

```
                                        + (a + b * d - e / f));
        }
}
```

Salida esperada:

```
a+b/d = 20
a+b*d-e/f = 219
```

2. Usando operadores dentro de ( ): Cuando use el operador + dentro de System.out.println(), asegúrese de hacer la suma usando paréntesis. Si escribimos algo antes de hacer la suma, entonces se lleva a cabo la suma de cadenas, es decir, la asociatividad de la suma es de izquierda a derecha y, por lo tanto, los enteros se agregan a una cadena produciendo primero una cadena, y los objetos de cadena se concatenan cuando se usa +. Por lo tanto, puede crear resultados no deseados.

```
public class Operadores {
        public static void main(String[] args)
        {

                int x = 5, y = 8;

                // concatena x y y como
                // primero se agrega x a la concatenación (x+y)=
                // produciendo “concatenacion(x+y)=5”
                // y entonces 8 se añade después.
                System.out.println("Concatenacion (x+y)= " + x + y);

                // adición de x y y.
                System.out.println("Suma (x+y) = " + (x + y));
        }
}
```

Salida esperada:

```
Concatenacion (x+y)= 58
Suma (x+y) = 13
```

Ejemplo con operadores aritméticos

```
import java.util.Scanner;
```

```
public class OperadoresAritmeticos {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);

		System.out.print("Agrega el primer numero: ");
		double num1 = sc.nextDouble();

		System.out.print("Agrega el segundo numero: ");
		double num2 = sc.nextDouble();

		double sum = num1 + num2;
		double difference = num1 - num2;
		double product = num1 * num2;
		double quotient = num1 / num2;

		System.out.println("La suma de dos numeros es: " + sum);
		System.out.println("La resta de dos numeros es: " + difference);
		System.out.println("El producto de dos numeros es: " + product);
		System.out.println("La division de dos numeros es: " + quotient);
	}
}
```

Ejemplos con operadores de relación

Ejemplo =

```
// Código Java para el operador “=”

import java.io.*;

class Ejemplo {
	public static void main(String[] args)
	{
		// Declaración de variables
		int num;
		String name;

		// Asignación de valores
		num = 10;
		name = "Computacion";

		// Mostrando los valores aplicados
```

```
                System.out.println("num es asignado: " + num);
                System.out.println("name es asignado: " + name);
        }
}
```

Ejemplo +=

```
// Código Java para el operador "+="

import java.io.*;

class Ejemplo {
        public static void main(String[] args)
        {

                // Declaración de variables
                int num1 = 10, num2 = 20;

                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);

                // Sumando y asignando valores
                num1 += num2;

                // Mostrando los valores aplicados
                System.out.println("num1 = " + num1);
        }
}
```

Ejemplo -=

```
// Código Java para el operador "-="

import java.io.*;

class Ejemplo {
        public static void main(String[] args)
        {

                // Declaración de variables
                int num1 = 10, num2 = 20;

                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);
```

```
                // Restando & asignando valores
                num1 -= num2;

                // Mostrando los valores aplicados
                System.out.println("num1 = " + num1);
        }
}
```

Ejemplo *=

```
// Código Java para el operador "*="

import java.io.*;

class Ejemplo {
        public static void main(String[] args)
        {

                // Declaración de variables
                int num1 = 10, num2 = 20;

                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);

                // Multiplicando & asignando valores
                num1 *= num2;

                // Mostrando los valores aplicados
                System.out.println("num1 = " + num1);
        }
}
```

Ejemplo /=

```
// Código Java para el operador "/="

import java.io.*;

class Ejemplo {
        public static void main(String[] args)
        {

                // Declaración de variables
```

```
                int num1 = 20, num2 = 10;

                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);

                // Dividiendo & asignando valores
                num1 /= num2;

                // Mostrando los valores aplicados
                System.out.println("num1 = " + num1);
        }
}
```

Ejemplo %=

```
// Código Java para el operador "%="

import java.io.*;

class Ejemplo {
        public static void main(String[] args)
        {

                // Declaración de variables
                int num1 = 5, num2 = 3;

                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);

                // Modulando & asignando valores
                num1 %= num2;

                // Mostrando los valores aplicados
                System.out.println("num1 = " + num1);
        }
}
```

Ejemplo con operadores unarios

```
import java.util.Scanner;

public class OperadoresUnarios {
        public static void main(String[] args)
```

```
	{
		Scanner sc = new Scanner(System.in);

		int num = 10;

		int result = +num;
		System.out.println(
			"El valor del resultado despues de operador unario mas es: "
			+ result);

		result = -num;
		System.out.println(
			"El valor del resultado despues de operador unario menos es: "
			+ result);

		result = ++num;
		System.out.println(
			"El valor del resultado despues del pre-incremento es: "
			+ result);

		result = num++;
		System.out.println(
			"El valor del resultado despues del post-incremento es: "
			+ result);

		result = --num;
		System.out.println(
			"El valor del resultado despues del pre-decremento es: "
			+ result);

		result = num--;
		System.out.println(
			"El valor del resultado despues del post-decremento es: "
			+ result);
	}
}
```

Ejemplo con operadores de relación

```
import java.util.Scanner;

public class OperadoresRelacionales {
public static void main(String[] args) {
```

```
        Scanner scan = new Scanner(System.in);

        System.out.println("Agrega el primer numero: ");
        int num1 = scan.nextInt();

        System.out.println("Agrega el segundo numero: ");
        int num2 = scan.nextInt();

        int num1 =1;
        int num2 = 2;


        System.out.println("num1 > num2 es " + (num1 > num2));
        System.out.println("num1 < num2 es " + (num1 < num2));
        System.out.println("num1 >= num2 es " + (num1 >= num2));
        System.out.println("num1 <= num2 es " + (num1 <= num2));
        System.out.println("num1 == num2 es " + (num1 == num2));
        System.out.println("num1 != num2 es " + (num1 != num2));
}
}
```

Ejemplo con operadores lógicos

```
public class OperadoresLogicos {
        public static void main(String[] args) {
                boolean a = true;
                boolean b = false;

                System.out.println("a: " + a);
                System.out.println("b: " + b);
                System.out.println("a && b: " + (a && b));
                System.out.println("a || b: " + (a || b));
                System.out.println("!a: " + !a);
                System.out.println("!b: " + !b);
        }
}
```

Control de Flujo en Java

Toma de decisiones

La toma de decisiones en la programación es similar a la toma de decisiones en la vida real. En programación también nos enfrentamos a algunas situaciones en las que queremos que un determinado bloque de código se ejecute cuando se cumpla alguna condición.

Un lenguaje de programación utiliza declaraciones de control para controlar el flujo de ejecución de un programa en función de ciertas condiciones. Estos se utilizan para hacer que el flujo de ejecución avance y se bifurque en función de los cambios en el estado de un programa.

Sentencia if

Es la declaración de toma de decisiones más simple. Se utiliza para decidir si una determinada instrucción o bloque de instrucciones se ejecutará o no, es decir, si una determinada condición es verdadera, entonces se ejecutará un bloque de instrucciones; de lo contrario, no se ejecutará.

Sintaxis

```
if(condición)
{
   // Sentencias que se ejecutan si
   // condición es verdadera
}
```

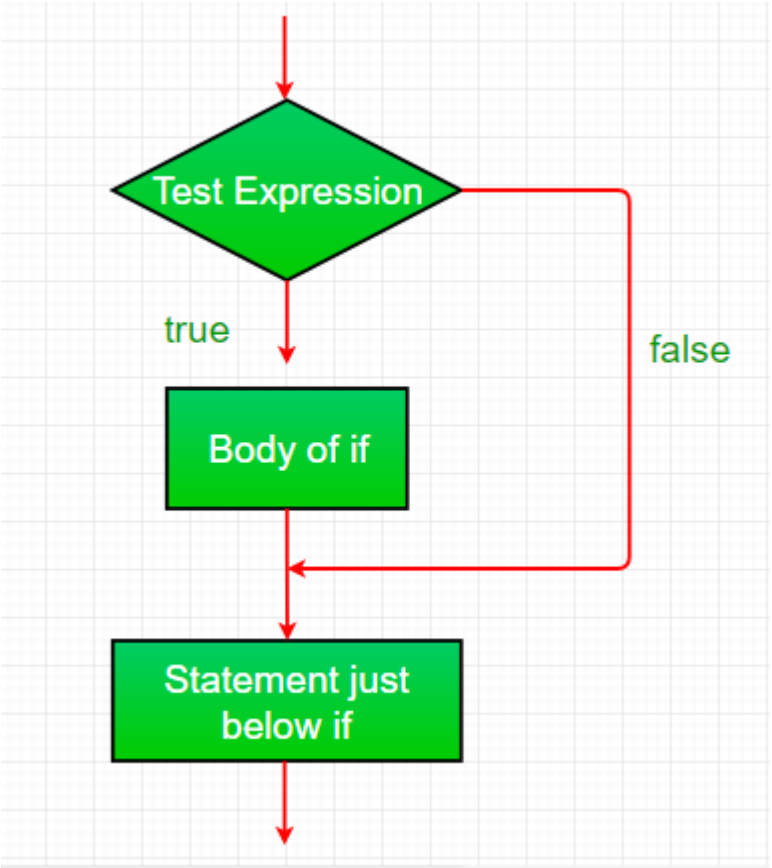
Aquí, la **condición** después de la evaluación será verdadera o falsa. Si la declaración acepta valores booleanos: si el valor es verdadero, ejecutará el bloque de declaraciones debajo de él.
Si no proporcionamos las llaves '{' y '}' después de **if (condición)**, entonces, por defecto, la declaración if considerará que la declaración inmediata está dentro de su bloque. Por ejemplo,

```
if(condición)  //Se asume la condición como verdadera
   sentencia1;  //parte del bloque if
   sentencia2;  //aparte del bloque if

// Aquí si la condición es verdadera
// El bloque if se considera una sentencia1 es parte y se ejecuta como una única condición
```

// La sentencia2 estará separada del bloque if entonces se ejecuta independiente si la condición es verdadero o falso.

Diagrama de flujo



Ejemplos:

```
// Programa Java para bloque if

class IfDemo {
	public static void main(String args[])
	{
		String str = "Computacion";
		int i = 4;

		// if block
		if (i == 4) {
			i++;
			System.out.println(str);
		}

		// Ejecutado por defecto
		System.out.println("i = " + i);
```

```
	}
}

public class IfElseExample {
	public static void main(String[] args) {
		boolean a = true;
		boolean b = false;

		if (a) {
			System.out.println("a es verdadero");
		} else {
			System.out.println("a es falso");
		}

		if (b) {
			System.out.println("b es verdadero");
		} else {
			System.out.println("b es falso");
		}
	}
}
```

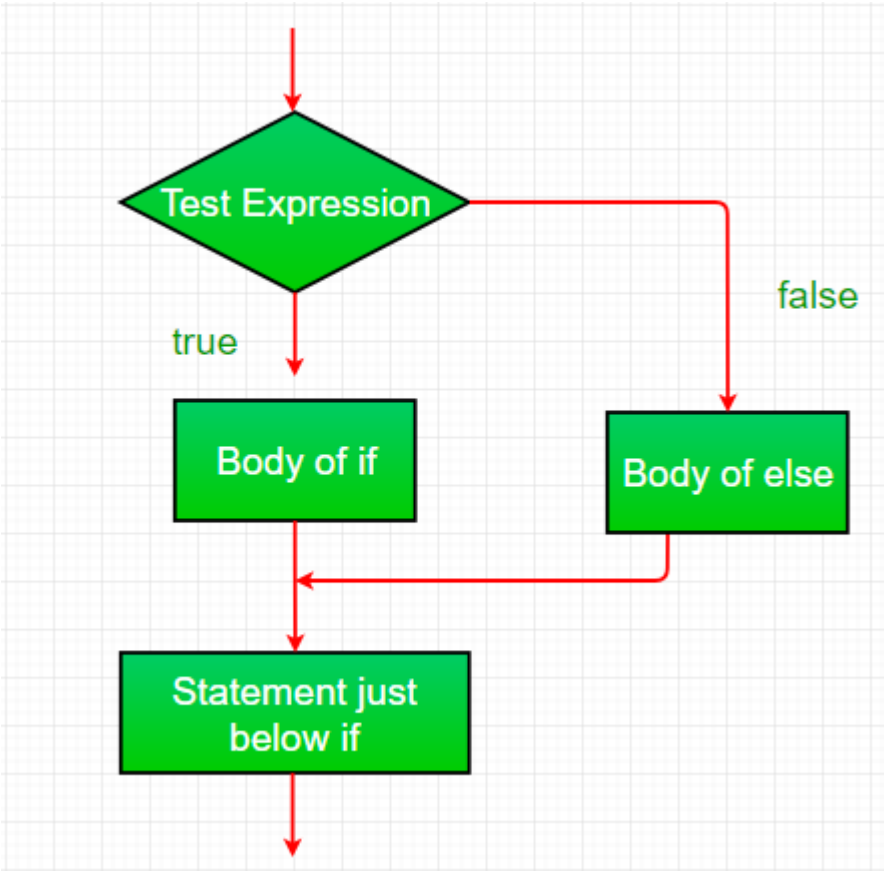
Bloque if-else

La declaración if por sí sola nos dice que si una condición es verdadera ejecutará un bloque de declaraciones y si la condición es falsa no lo hará. Pero, ¿y si queremos hacer otra cosa si la condición es falsa? Aquí viene la declaración else. Podemos usar la declaración else con la declaración if para ejecutar un bloque de código cuando la condición es falsa.

Sintaxis

```
if (condición)
{
   // Ejecuta el bloque if
   // si la condición es true
}
else
{
   // Ejecuta el bloque if
   // si la condición es false
}
```

Diagrama de flujo



Ejemplo:

```
// Ejemplo if-else

class IfElseDemo {
        public static void main(String args[])
        {
                int i = 20;

                if (i < 15)
                        System.out.println("i es mas pequeño que 15");
                else
                        System.out.println("i es mas grande que 15");

                System.out.println("Fuera del bloque if-else");
        }
}
```

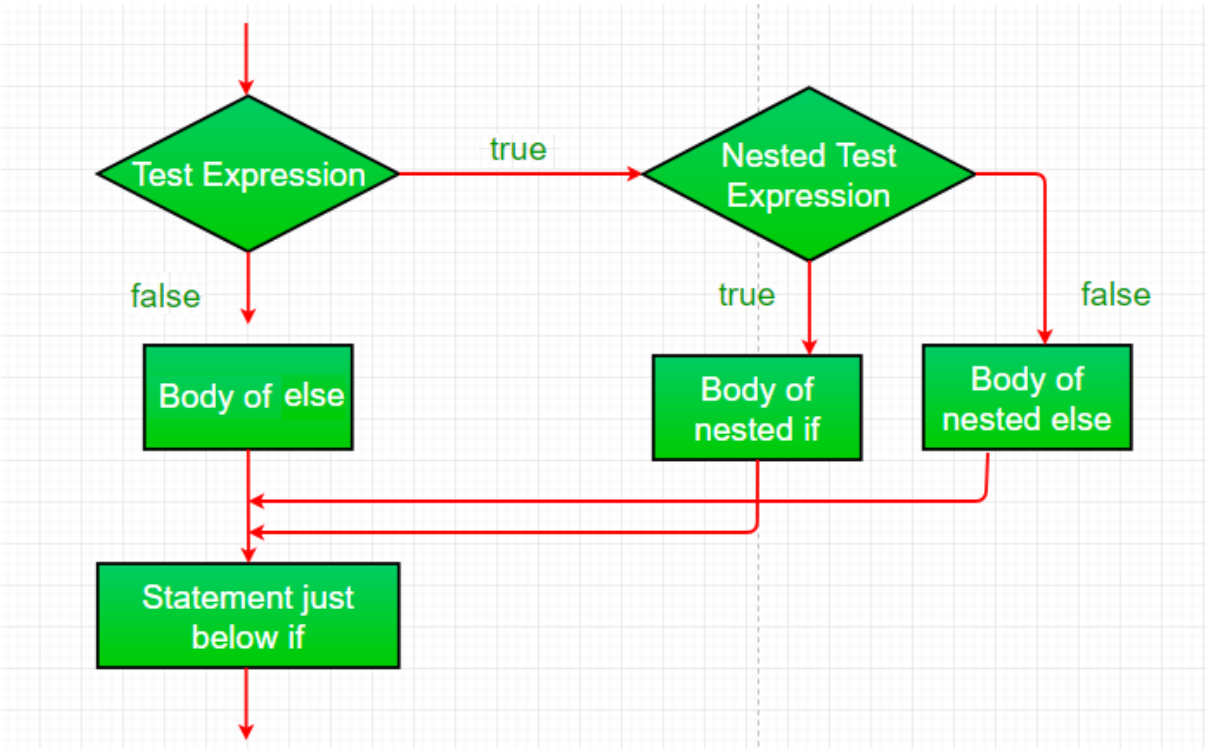
If anidado

Un if anidado es una declaración if que es el destino de otro if o else. Las declaraciones if anidadas significan una declaración if dentro de una declaración if. Sí, java nos permite anidar sentencias if dentro de sentencias if. es decir, podemos colocar una sentencia if dentro de otra sentencia if.

Sintaxis

```
if (condición1)
{
  // Se ejecuta cuando condición1 es true
  if (condición2)
  {
    // Se ejecuta cuando condición2 es true
  }
}
```

Diagrama de flujo



Bloque if else-if

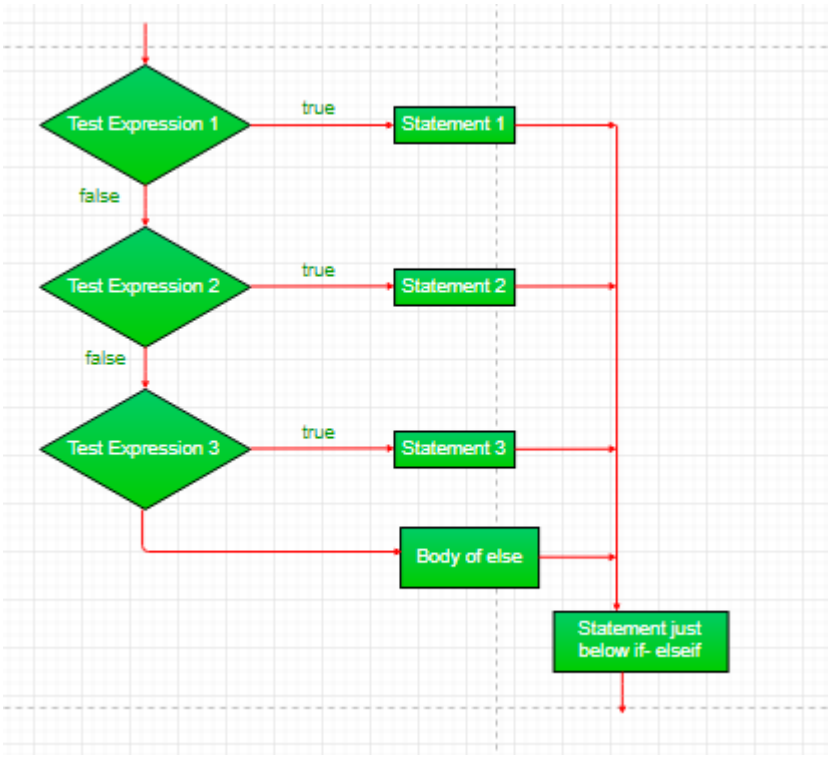
Aquí, un usuario puede decidir entre múltiples opciones. Las declaraciones if se ejecutan de arriba hacia abajo. Tan pronto como una de las condiciones que controlan el si es verdadera, se ejecuta la declaración asociada con ese "si" y se

omite el resto de la escalera. Si ninguna de las condiciones es verdadera, entonces se ejecutará la instrucción else final. Puede haber tantos como bloques "si no" asociados con un bloque "si", pero solo se permite un bloque "si no" con un bloque "si".

Sintaxis

```
if (condición)
   sentencia;
else if (condición)
   sentencia;
.
.
else
   sentencia;
```

Diagrama de flujo



Ejemplo:

```
// Programa Java para ilustrar escalera if-else-if

import java.io.*;

class Ejemplo {
        public static void main(String[] args)
        {
```

```
                // iniciando expresión
                int i = 20;

                // condición 1
                if (i < 10)
                        System.out.println("i es menor que 10\n");

                // condition 2
                else if (i < 15)
                        System.out.println("i es menor que 15\n");

                // condition 3
                else if (i < 20)
                        System.out.println("i es menor que 20\n");

                else
                        System.out.println("i es mayor que "
                                                + "o igual a 20\n");

                System.out.println("Fuera del bloque if-else-if");
        }
}
```

Bloque Switch

La declaración de cambio es una declaración de bifurcación de múltiples vías. Proporciona una forma sencilla de enviar la ejecución a diferentes partes del código en función del valor de la expresión.

Sintaxis

```
switch (expresión)
{
  case valor1:
    sentencia1;
    break;
  case valor2:
    sentencia2;
    break;
  .
  .
  case valorN:
```

```
    sentenciaN;
    break;
  default:
    sentenciaDefault;
}
```

Ejemplo

```
import java.io.*;

class Ejemplo {
        public static void main (String[] args) {
                int num=20;
                switch(num){
                case 5 : System.out.println("Es 5");
                                        break;
                case 10 : System.out.println("Es 10");
                                        break;
                case 15 : System.out.println("Es 15");
                                        break;
                case 20 : System.out.println("Es 20");
                                        break;
                default: System.out.println("No esta presente");

                }
        }
}
```

Ciclos

El bucle en los lenguajes de programación es una característica que facilita la ejecución de un conjunto de instrucciones/funciones repetidamente mientras alguna condición se evalúa como verdadera. Java proporciona formas de ejecutar los bucles. Si bien todas las formas brindan una funcionalidad básica similar, difieren en su sintaxis y tiempo de verificación de condición.

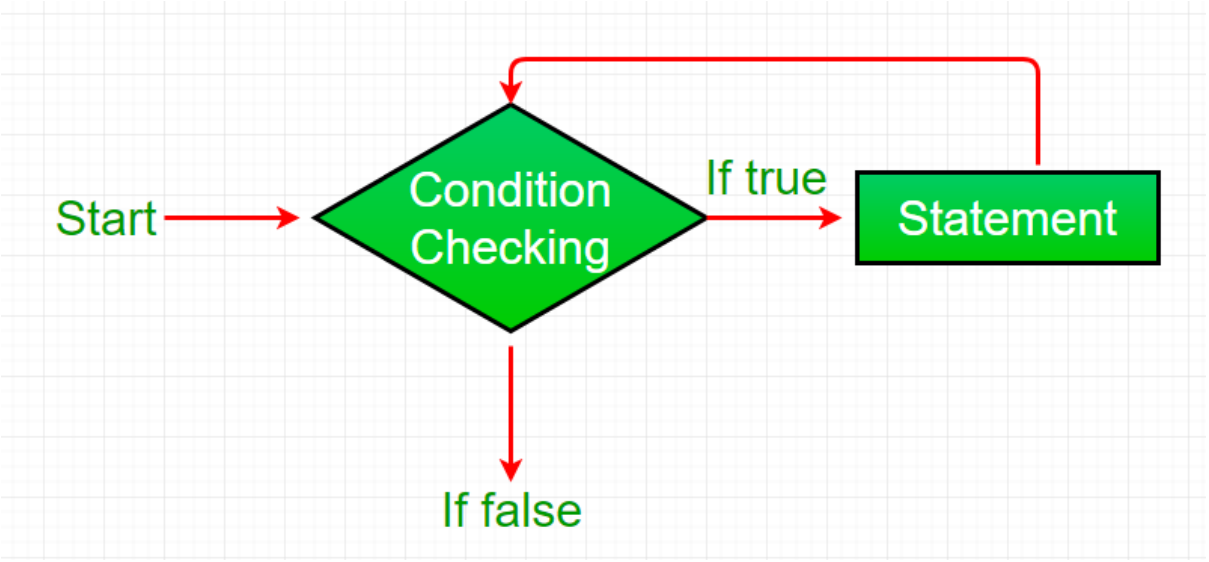
Ciclo While

Un ciclo while es una declaración de flujo de control que permite que el código se ejecute repetidamente en función de una condición booleana determinada. El ciclo while se puede considerar como una declaración if repetida.

Sintaxis

```
while (condición booleana)
{
  sentencias del ciclo...
}
```

Diagrama de flujo



- Mientras que el ciclo comienza con la verificación de la condición booleana. Si se evaluó como verdadero, entonces se ejecutan las declaraciones del cuerpo del ciclo; de lo contrario, se ejecuta la primera declaración que sigue al ciclo. Por este motivo también se denomina **bucle de control de entrada**.
- Una vez que la condición se evalúa como verdadera, se ejecutan las declaraciones en el cuerpo del ciclo. Normalmente, las declaraciones contienen un valor de actualización para la variable que se procesa para la siguiente iteración.
- Cuando la condición se vuelve falsa, el ciclo termina, lo que marca el final de su ciclo de vida.

Ejemplo:

```
import java.io.*;

class Ejemplo {
	public static void main (String[] args) {
	int i=0;
	while (i<=10)
	{
		System.out.println(i);
		i++;
	}
```

```
        }
}
```

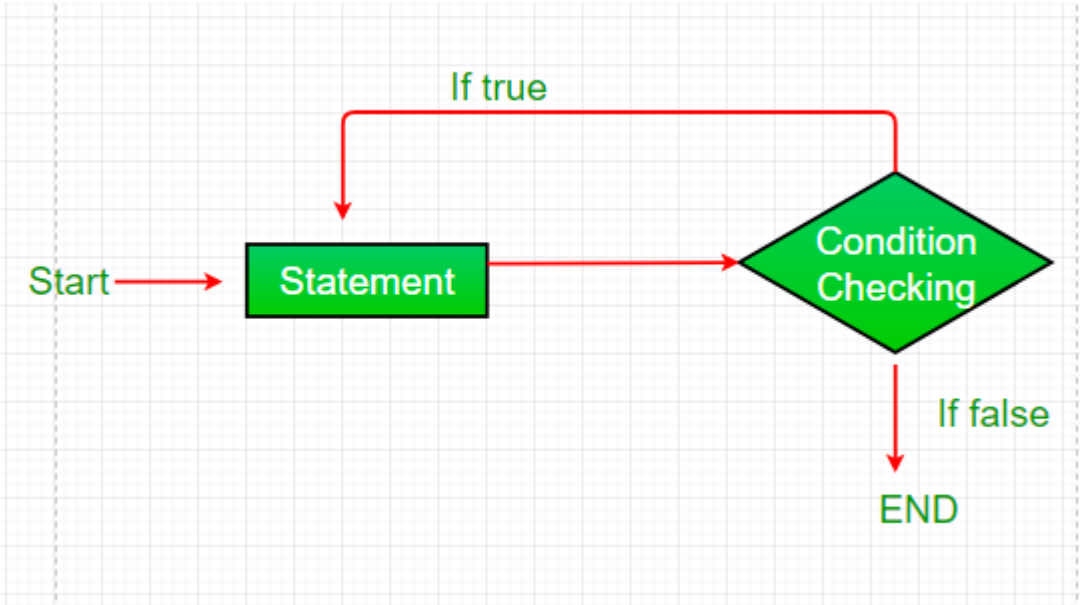
Ciclo do-while

El ciclo do while es similar a while loop con la única diferencia de que verifica la condición después de ejecutar las declaraciones y, por lo tanto, es un ejemplo de **Exit Control Loop**.

Sintaxis

```
do
{
   sentencias...
}
while (condición);
```

Diagrama de flujo



- El ciclo do while comienza con la ejecución de la(s) sentencia(s). No se verifica ninguna condición por primera vez.
- Después de la ejecución de las sentencias y la actualización del valor de la variable, se comprueba si la condición tiene un valor verdadero o falso. Si se evalúa como verdadero, comienza la siguiente iteración del ciclo.
- Cuando la condición se vuelve falsa, el ciclo termina, lo que marca el final de su ciclo de vida.
- Es importante tener en cuenta que el ciclo do-while ejecutará sus declaraciones al menos una vez antes de que se verifique cualquier condición y, por lo tanto, es un ejemplo de ciclo de control de salida.

Ejemplo:

```
import java.io.*;

class Ejemplo {
        public static void main (String[] args) {
        int i=0;
        do
        {
                System.out.println(i);
                i++;
        }while(i<=10);
        }
}
```

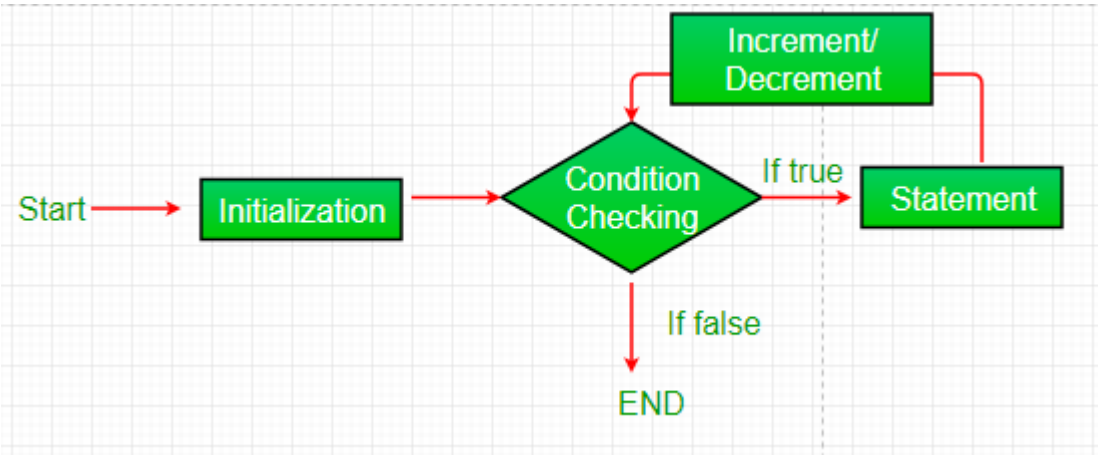
Ciclo For

El ciclo for proporciona una forma concisa de escribir la estructura del bucle. A diferencia de un bucle while, una instrucción for consume la inicialización, la condición y el incremento/decremento en una línea, lo que proporciona una estructura de bucle más corta y fácil de depurar.

Sintaxis

```
for (condición de inicialización; condición de prueba; incremento/decremento)
{
   sentencias
}
```

Diagrama de flujo



- **Condición de inicialización:** Aquí, inicializamos la variable en uso. Marca el inicio de un bucle for. Se puede usar una variable ya declarada o se puede declarar una variable, solo local para bucle.
- **Condición de prueba:** se utiliza para probar la condición de salida de un bucle. Debe devolver un valor booleano. También es un **bucle de control de entrada**, ya que la condición se comprueba antes de la ejecución de las instrucciones del bucle.
- **Ejecución de declaraciones:** una vez que la condición se evalúa como verdadera, se ejecutan las declaraciones en el cuerpo del ciclo.
- **Incremento/Decremento:** Se utiliza para actualizar la variable para la siguiente iteración.
- **Terminación del bucle:** Cuando la condición se vuelve falsa, el bucle termina marcando el final de su ciclo de vida.

Ejemplo:

```
import java.io.*;

class Ejemplo {
	public static void main (String[] args) {
	for (int i=0;i<=10;i++)
	{
		System.out.println(i);
	}
	}
}
```

Trampas con los ciclos

Ciclo infinito

Uno de los errores más comunes al implementar cualquier tipo de bucle es que puede que nunca salga, es decir, el bucle se ejecuta durante un tiempo infinito. Esto sucede cuando la condición falla por alguna razón.

```
import java.io.*;

class Ejemplo {
	public static void main(String[] args)
	{
		for (;;) {
		}
	}
```

```
}

import java.io.*;

class Ejemplo {
	public static void main (String[] args) {
	while (true)
	{
		// sentencias
	}

	}
}

public class Ejemplo
{
	public static void main(String[] args)
	{

		for (int i = 5; i != 0; i -= 2)
		{
			System.out.println(i);
		}
		int x = 5;

		while (x == 5)
		{
			System.out.println("Dentro del ciclo");
		}
	}
}
```

Ciclo anidado

```
import java.io.*;

class Ejemplo {
	public static void main (String[] args) {
		for(int i = 0; i < 3; i++){
		for(int j = 0; j < 2; j++){
			System.out.println(i);
		}
		System.out.println();
		}
```

```
	}
}

import java.io.*;

class Ejemplo {
	public static void main(String[] args)
	{
		int i = 1, j = 1;
		while (i <= 3) {
			while (j <= 3) {
				System.out.print(j);
				j++;
			}
			i++;
			System.out.println("");
			j = 1;
		}
	}
}

import java.io.*;

class Ejemplo {
	public static void main(String[] args)
	{

		int row = 1, column = 1;
		int x;
		do {
			x = 4;
			do {
				System.out.print("");
				x--;
			} while (x >= row);
			column = 1;
			do {
				System.out.print(column + " ");
				column++;

			} while (column <= 5);
			System.out.println(" ");
			row++;
		} while (row <= 5);
```

```
	}
}

import java.io.*;

class Ejemplo {
	public static void main(String[] args)
	{

		int weeks = 3;
		int days = 7;
		int i = 1;

		while (i <= weeks) {
			System.out.println("Semana: " + i);

			// ciclo interno
			for (int j = 1; j <= days; ++j) {
				System.out.println(" Dias:: " + j);
			}
			++i;
		}
	}
}
```