## Operadores lógicos

|  |  |  |  |
| --- | --- | --- | --- |
| **Operador** | **Descripción** | **Uso** | **Retorna cierto si** |
| && | and | op1 && op2 | op1 y op2 son true. Solo evalúa op2 si op1 es true |
| || | or | op1 || op2 | op1 o op2 es true. Solo evalúa op2 si op1 es false |
| ! | not | !op | op es false |

## Operadores lógicos a nivel de bits

|  |  |  |
| --- | --- | --- |
| **Operador** | **Descripción** | **Uso** |
| & | and | op1 & op2 |
| | | or | op1 | op2 |
| ^ | or exclusivo | op1 ^ op2 |
| ~ | complemento | ~op1 |

El operador **AND** **bit por bit (&)** realiza un and lógico entre cada par de bits paralelos que forman los operandos. Si ambos bits valen 1 el bit resultante vale 1, de lo contrario vale 0. Por ejemplo:

|  |
| --- |
| int bits\_orig = 6; --------// binario 00000000 00000000 00000000 00000110 |
| int bits\_bandera = 5; -----// binario 00000000 00000000 00000000 00000101 |
| int respuesta = (bits\_orig & bits\_bandera);--------- --// 000000 00000100 |

El operador **OR bit por bit (|)** realiza un or lógico entre cada par de bits paralelos que forman los operandos. Si alguno de los bits vale 1 el bit resultante vale 1, de lo contrario vale 0. Por ejemplo:

|  |
| --- |
| int bits\_orig = 6; --------// binario 00000000 00000000 00000000 00000110 |
| int bits\_bandera = 5; -----// binario 00000000 00000000 00000000 00000101 |
| int respuesta = (bits\_orig | bits\_bandera);--------- --// 000000 00000111 |

El operador **XOR bit por bit (^)** realiza un or exclusivo entre cada par de bits paralelos que forman los operandos. Si ambos bits tienen valores opuestos el bit resultante vale 1, de lo contrario vale 0. Por ejemplo:

|  |
| --- |
| int bits\_orig = 6; --------// binario 00000000 00000000 00000000 00000110 |
| int bits\_bandera = 5; -----// binario 00000000 00000000 00000000 00000101 |
| int respuesta = (bits\_orig ^ bits\_bandera);--------- --// 000000 00000011 |

El operador **NOT bit por bit (~)** invierte los bits del operando. Por ejemplo:

|  |
| --- |
| byte valor = 6;-------------------// binario 00000110 |
| byte respuesta = (~ valor);---- --// binario 11111001 |

**Operadores de desplazamiento de bits**

Existen muchas operaciones que requieren desplazar los valores de los bits que componen un número a la izquierda o a la derecha. Los operadores de desplazamiento de bits suelen utilizarse para llevar a cabo operaciones muy rápidas de multiplicación y de división de enteros. Un desplazamiento a la izquierda equivale a una multiplicación por 2 y un desplazamiento a la derecha a una división por 2.

Un desplazamiento no es una rotación. A medida que se desplazan los bits hacia un extremo se van rellenando con ceros por el otro extremo. Los bits que salen se pierden.

|  |  |  |
| --- | --- | --- |
| **Operador** | **Uso** | **Operación** |
|  |  |  |
| << | op1 << op2 | Desplaza a la izquierda los bits del primer operando op1 tantas veces como indica el segundo operando op2( por la derecha siempre entra un cero). |
| >> | op1 >> op2 | Desplaza a la derecha los bits del primer operando op1 tantas veces como indica el segundo operando op2( por la izquierda entra siempre el bit más significativo anterior). |
| >>> | op1 >>> op2 | Desplaza a la derecha los bits del primer operando op1 tantas veces como indica el segundo operando op2(sin signo- por la izquierda entra siempre un cero) |

A diferencia de C y C++, Java siempre conserva el bit de signo (el bit izquierdo) después de hacer un desplazamiento. Este tipo de desplazamiento se conoce como desplazamiento aritmético o desplazamiento de extensión de signo. Las siguientes instrucciones desplazan un valor, primero dos bits a la izquierda y luego dos bits a la derecha, con los operadores de desplazamiento de bits de Java:

|  |
| --- |
| int original = -3; ----------------//---------------------- binario 10000000 00000000 00000000 00000011 |
| int izquierda = original << 2; ----// izquierda vale -12 ---binario 10000000 00000000 00000000 00001100 |
| int derecha = izquierda >> 2;----- // derecha vale -3 --- --binario 10000000 00000000 00000000 00000011 |
|  |
| int x = 7; -----------------------// x vale 7-------------- binario 00000000 00000000 00000000 00000111 |
| x = x << 1;-----------------------// x vale 14------------ -binario 00000000 00000000 00000000 00001110 |

El **operador >>>** realiza un desplazamiento a la derecha similar al operador >>. La diferencia es que el operador >>> no conserva el bit de signo, sino que coloca el bit de signo en cero, a diferencia del operador >>, que lo pone en uno cuando es negativo, de forma que permanezca negativo. Normalmente se utiliza el operador >>> cuando el operador no representa una cantidad con signo, sino que es una máscara de bits de algún tipo en el que no importa el bit de signo. El siguiente ejemplo muestra el uso del operador >>> para desplazar el valor 256 dos bits a la derecha, dando como resultado 64:

|  |
| --- |
| int mascara = 256;------------------// mascara 256-------------- binario 00000000 00000000 00000001 00000000 |
| int respuesta = mascara >>> 2;------// respuesta 64------------- binario 00000000 00000000 00000000 01000000 |

**Nota:**

Java convierte a tipo int los tipo short y byte antes de realizar el desplazamiento de bits. Esto significa que el tipo byte tendrá 32 bits en lugar de 8. Después de ejecutar el desplazamiento trunca el dato para convertirlo al tipo original.