

Operators

Arithmetic operators

Operator	Result
+	Addition
-	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement

The Modulus Operator

- The modulus operator, %, can be applied to floating-point types as well as integer types

```
// Demonstrate the % operator.  
class Modulus {  
    public static void main(String args[]) {  
        int x = 42;  
        double y = 42.25;  
  
        System.out.println("x mod 10 = " + x % 10);  
        System.out.println("y mod 10 = " + y % 10);  
    }  
}
```

The Bitwise Operators

- Java defines several bitwise operators that can be applied to the integer types, long, int, short, char, and byte.
- These operators act upon the individual bits of their operands
- Note: Java uses two's complement to represent the negative numbers

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Left Shift operator

- The left shift operator, `<<`, shifts all of the bits in a value to the left a specified number of times.
 - `value << num`
- For each shift left, the high-order bit is shifted out (and lost), and a zero is brought in on the right
- Java's may produce unexpected results when you are shifting **byte** and **short** values.

Example

```
class ByteShift {  
    public static void main(String args[]) {  
        byte a = 64, b;  
        int i;  
  
        i = a << 2;  
        b = (byte) (a << 2);  
  
        System.out.println("Original value of a: " + a);  
        System.out.println("i and b: " + i + " " + b);  
    }  
}
```

Right shift

- The right shift operator, `>>`, shifts all of the bits in a value to the right a specified number of times.
 - `value >> num`
 - `00100011` (35) when applied `>> 2` results in `00001000` (8)
- During shifting right, the top (leftmost) bits exposed by the right shift are filled in with the previous contents of the top bit.
- This is called sign extension. It aims to preserve the sign of negative numbers
 - `11111000` (-8) when applied `>>1` results in `11111100` (-4)

The Unsigned Right Shift

- We mostly want to insert a zero into the high-order bit no matter what its initial value was.
- This is called an unsigned shift.
- In java, the unsigned shift-right operator, `>>>`, always shifts zeros into the high-order bit.

11111111 11111111 11111111 11111111 (-1)

`>>> 24`

00000000 00000000 00000000 11111111 (255)

Example

// Unsigned shifting a byte value.

```
class ByteUShift {
    static public void main(String args[]) {
        char hex[] = {
            '0', '1', '2', '3', '4', '5', '6', '7',
            '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'
        };
        byte b = (byte) 0xf1;
        byte c = (byte) (b >> 4);
        byte d = (byte) (b >>> 4);
        byte e = (byte) ((b & 0xff) >> 4);

        System.out.println(" b = 0x"+ hex[(b >> 4) & 0x0f] + hex[b & 0x0f]);
        System.out.println(" b >> 4 = 0x"+ hex[(c >> 4) & 0x0f] + hex[c & 0x0f]);
        System.out.println(" b >>> 4 = 0x"+ hex[(d >> 4) & 0x0f] + hex[d & 0x0f]);
        System.out.println("(b & 0xff) >> 4 = 0x"+ hex[(e >> 4) & 0x0f] + hex[e & 0x0f]);
    }
}
```

Relational Operators

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Boolean Logical Operators

Operator	Result
&	Logical AND
	Logical OR
^	Logical XOR (exclusive OR)
	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

- The Boolean logical operators operate only on boolean operands.
- All of the binary logical operators combine two boolean values to form a resultant boolean value.


Short-Circuit Logical Operators

- The `&&` and `||`
- If you use the `||` and `&&` forms, Java will not bother to evaluate the right-hand operand when the outcome of the expression can be determined by the left operand alone.
- This is very useful when the right-hand operand depends on the value of the left one in order to function properly.

Task

- Read about
- Assignment operator (=)
- Ternary operator (?)

Java operators precedence table



• () [] .	HIGHEST
• ++ -- ~ !	
• * / %	
• + -	
• >> >>> <<	
• > >= < <=	
• == !=	
• &	
• ^	
•	
• &&	
•	
• ?:	
• = op=	LOWEST