Java Operators

The Java programming language has around 30 operators as summarized in the following table:

**Category Operators**

|  |  |
| --- | --- |
| Simple assignment | = |
| Arithmetic | + - \* / % |
| Unary | + - ++ -- ! |
| Relational | == != > >= < <= |
| Conditional | && || ? : (ternary) |
| Type comparison | instanceof |
| Bitwise and Bit shift | ~ << >> >>> & ^ | |

1. Simple assignment

Perhaps this is the most commonly used operator. It assigns the value on its right to the operand on its left. Here are some examples:

* + Assigning numbers:

|  |  |
| --- | --- |
| 1  2 | int x = 10;  float y = 3.5F; |

* + Assigning object references:

|  |  |
| --- | --- |
|  | String message = “Hello world”;  File csv = new File(“test.csv”); |

2. Arithmetic operators

The arithmetic operators are used to perform mathematic calculations just like basic mathematics in school. The following table lists all arithmetic operators in Java:

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| **+** | Addition (and strings concatenation) operator |
| **-** | Subtraction operator |
| **\*** | Multiplication operator |
| **/** | Division operator |
| **%** | Remainder operator |

Here’s an example program:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | public class ArithmeticDemo {      public static void main(String[] args) {          int x = 10;          int y = 20;            int result = x + y;            System.out.println("x + y = " + result);            result = x - y;            System.out.println("x - y = " + result);            result = x \* y;            System.out.println("x \* y = " + result);            result = y / x;            System.out.println("y / x = " + result);            result = x % 3;            System.out.println("x % 3 = " + result);        }  } |

Output:

|  |  |
| --- | --- |
| 1  2  3  4  5 | x + y = 30  x - y = -10  x \* y = 200  y / x = 2  x % 3 = 1 |

The addition operator (+) can also be used for joining two or more strings together (strings concatenation). Here’s an example program:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class StringConcatDemo {      public static void main(String[] args) {          String firstName = "James";          String lastName = "Gosling";          String greeting = "Hello " + firstName + " " + lastName;            System.out.println(greeting);      }  } |

Output:

|  |  |
| --- | --- |
|  | Hello James Gosling! |

3. Unary operators

The unary operators involve in only a single operand. The following table lists all unary operators in Java:

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| **+** | Unary plus operator; indicates positive value (numbers are positive by default, without this operator). |
| **-** | Unary minus operator; negate an expression. |
| **++** | Increment operator; increments a value by 1. |
| **--** | Decrement operator; decrements a value by 1; |
| **!** | Logical complement operator; inverts value of a boolean. |

Here’s an example program:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | public class UnaryDemo {      public static void main(String[] args) {          int x = 10;          int y = 20;            int result = +x;            System.out.println("+x = " + result);            result = -y;            System.out.println("-y = " + result);            result = ++x;            System.out.println("++x = " + result);            result = --y;            System.out.println("--y = " + result);            boolean ok = false;            System.out.println(ok);            System.out.println(!ok);      }  } |

Output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | +x = 10  -y = -20  ++x = 11  ++y = 19  false  true |

Note that the increment and decrement operators can be placed before (prefix) or after (postfix) the operand, e.g. ++x or x++, --y or y--. When using these two forms in an expression, the difference is:

* + Prefix form: the operand is incremented or decremented before used in the expression.
  + Postfix form: the operand is incremented or decremented after used in the expression.

The following example illustrates the prefix/postfix:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | public class PrefixPostfixDemo {      public static void main(String[] args) {          int x = 10;          int y = 20;            System.out.println(++x);            System.out.println(x++);            System.out.println(x);            System.out.println(--y);            System.out.println(y--);            System.out.println(y);      }  } |

Output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 11  11  12  19  19  18 |

4. Relational operators

The relational operators are used to compare two operands or two expressions and result is a boolean. The following table lists all relational operators in Java.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| **==** | equal to |
| **!=** | not equal to |
| **>** | greater than |
| **>=** | greater than or equal to |
| **<** | less than |
| **<=** | less than or equal to |

Here’s an example program:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | public class RelationalDemo {      public static void main(String[] args) {          int x = 10;          int y = 20;            boolean result = x == y;            System.out.println("x == y? " + result);            result = x != y;            System.out.println("x != y? " + result);            result = x > y;            System.out.println("x > y? " + result);            result = x >= y;            System.out.println("x >= y? " + result);            result = x < y;            System.out.println("x < y? " + result);            result = x <= y;            System.out.println("x <= y? " + result);      }  } |

Output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | x == y? false  x != y? true  x > y? false  x >= y? false  x < y? true  x <= y? true |

5. Conditional operators

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| **&&** | conditional -AND operator |
| **||** | conditional-OR operator |
| **? :** | ternary operator in form of: *A ? B : C* |

The conditional operators (&& and ||) are used to perform conditional-AND and conditional-OR operations on two boolean expressions and result in a boolean value. They have “short-circuiting” behavior:

* + For the && operator: if the left expression is evaluated to false, then the right expression is not evaluated. Final result is false.
  + For the || operator: if the left expression is evaluated to true, then the right expression is not evaluated. Final result is true.

Here’s an example program:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public class ConditionalDemo {      public static void main(String[] args) {          int x = 10;          int y = 20;            if ((x > 8) && (y > 8)) {              System.out.println("Both x and y are greater than 8");          }            if ((x > 10) || (y > 10)) {              System.out.println("Either x or y is greater than 10");          }      }  } |

Output:

|  |  |
| --- | --- |
| 1  2 | Both x and y are less than 8  Either x or y is greater than 10 |

Other conditional operators are ? and : which form a ternary (three operands) in the following form:

**result = A ? B : C**

This is interpreted like this: if A evaluates to true, then evaluates B and assign its value to the result. Otherwise, if A evaluates to false, then evaluates C and assign its value to the result. For short, we can say: *If A then B else C*. So this is also referred as shorthand for an if-then-else statement.

Here’s an example of the ternary operator:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public class TernaryDemo {      public static void main(String[] args) {          int x = 10;          int y = 20;            int result = (x > 10) ? x : y;            System.out.println("result 1 is: " + result);            result = (y > 10) ? x : y;            System.out.println("result 2 is: " + result);        }  } |

Output:

|  |  |
| --- | --- |
| 1  2 | result 1 is: 20  result 2 is: 10 |

6. Type comparison operator (instanceof)

The instanceof operator tests if an object is an instance of a class, a subclass or a class that implements an interface; and result in a boolean value.

Here’s an example program:

|  |  |
| --- | --- |
|  | public class InstanceofDemo {      public static void main(String[] args) {          String name = "Java";           if (name instanceof String) {              System.out.println("an instance of String class");          }           // test for subclass of Object          if (name instanceof Object) {              System.out.println("an instance of Object class");          }           // test for subclass of an interface          if (name instanceof CharSequence) {              System.out.println("an instance of CharSequence interface");          }      }  } |

Output:

|  |  |
| --- | --- |
|  | an instance of String class  an instance of Object class  an instance of CharSequence interface |

7. Bitwise and Bit shift operators

These operators perform bitwise and bit shift operations on only integral types, not float types. They are rarely used so the listing here is just for reference:

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| **~** | unary bitwise complement; inverts a bit pattern |
| **<<** | signed left shift |
| **>>** | signed right shift |
| **>>>** | unsigned right shift |
| **&** | bitwise AND |
| **^** | bitwise exclusive OR |
| **|** | bitwise inclusive OR |

Here’s an example program:

|  |  |
| --- | --- |
|  | public class BitDemo {      public static void main(String[] args) {          int x = 10;          int result = x << 2;           System.out.println("Before left shift: " + x);          System.out.println("After left shift: " + result);      }  } |

Output:

|  |  |
| --- | --- |
|  | Before left shift: 10  After left shift: 40 |