Rules for Java identifiers:

* An identifier must start with a letter, an underscore (\_), or a dollar sign ($).
* Subsequent characters can be letters, digits, underscores, or dollar signs.
* Identifiers are case-sensitive.
* Identifiers cannot be keywords.

| Names can contain letters, digits, underscores, and dollar signs |
| --- |
| Names must begin with a letter |
| Names should start with a lowercase letter and it cannot contain whitespace |
| Names can also begin with $ and \_ |
| Names are case sensitive ("myVar" and "myvar" are different variables) |
| Reserved words (like Java keywords, such as int or boolean) cannot be used as names |

Examples of valid Java identifiers:

age\_var\_1

sum

Sum

totalVolume

int\_variable

float\_variable

Examples of invalid Java identifiers:

123\_variable (identifiers cannot start with a number)

my-variable (identifiers cannot contain hyphens)

class (class is a keyword)

It is generally considered good practice to use descriptive names for identifiers, and to avoid using abbreviations or acronyms unless they are well-known.

Operators in Java perform operations on variables and values. There are several types of operators, including:

* Arithmetic operators: Perform common mathematical operations:
  + + Addition
  + - Subtraction
  + \* Multiplication
  + / Division
  + % Remainder or Modulo
* Assignment operators: Assign values to variables:
  + = Assigns a value to a variable
* Comparison operators: Compare two values or variables:
  + **==, Equal to** returns true if the left-hand side is equal to the right-hand side.
  + **!=, Not Equal to** returns true if the left-hand side is not equal to the right-hand side.
  + **<, less than:** returns true if the left-hand side is less than the right-hand side.
  + **<=, less than or equal to** returns true if the left-hand side is less than or equal to the right-hand side.
  + **>, Greater than:** returns true if the left-hand side is greater than the right-hand side.
  + **>=, Greater than or equal to** returns true if the left-hand side is greater than or equal to the right-hand side.
* Logical operators: Test for true or false values:
  + && Logical AND
  + || Logical OR
  + ! Logical NOT
* Unary operators: Perform operations on a single value - Unary operators need only one operand. They are used to increment, decrement, or negate a value.
  + **– :** **Unary minus**, used for negating the values.
  + **+ :** **Unary plus** indicates the positive value (numbers are positive without this, however). It performs an automatic conversion to int when the type of its operand is the byte, char, or short. This is called unary numeric promotion.
  + **++ :** **Increment operator**, used for incrementing the value by 1. There are two varieties of increment operators.   
    - **Post-Increment:** Value is first used for computing the result and then incremented.
    - **Pre-Increment:** Value is incremented first, and then the result is computed.
  + **– – : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operators.   
    - **Post-decrement:** Value is first used for computing the result and then decremented.
    - **Pre-Decrement: The value** is decremented first, and then the result is computed.
  + **! : Logical not operator**, used for inverting a boolean value.
* Bitwise operators: Perform operations on bits:
  + **&, Bitwise AND operator:** returns bit by bit AND of input values.
  + **|, Bitwise OR operator:** returns bit by bit OR of input values.
  + **^, Bitwise XOR operator:** returns bit-by-bit XOR of input values.
  + **~, Bitwise Complement Operator:** This is a unary operator which returns the one’s complement representation of the input value, i.e., with all bits inverted.
* Ternary operators: Perform a conditional operation - The ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name Ternary.
  + condition **?** if true **:** if false
* Shift operators: Shift bits left or right:
  + << Left shift
  + >> Right shift
  + >>> Unsigned right shift

Here are some examples of how to use operators in Java:

Java

*// Arithmetic operators*

int a = 10;

int b = 5;

int c = a + b; *// c is now 15*

*// Assignment operators*

int d = 20;

c = d; *// c is now 20*

*// Comparison operators*

boolean e = a == b; *// e is now false*

boolean f = a != b; *// f is now true*

*// Logical operators*

boolean g = a < b && b < c; *// g is now false*

boolean h = a > b || b > c; *// h is now true*

*// Unary operators*

int i = -a; *// i is now -10*

int j = ++a; *// j is now 11, a is now 11*

*// Bitwise operators*

int k = a & b; *// k is now 0*

int l = a | b; *// l is now 15*

*// Ternary operators*

int m = a > b ? a : b; *// m is now 10*

*// Shift operators*

int n = a << 2; *// n is now 40*

int o = a >> 2; // o is now 2