Operators

Operators in Java are symbols that perform **operations** on one or more **operands** (values or variables) to produce a result. Java has a wide range of operators, including **arithmetic, comparison, logical, bitwise**, and **assignment operators**.

#Arithmetic Operators

Arithmetic operators are a fundamental part of any programming language and are commonly used in mathematical expressions and equations in Java programs. You can write more effective and efficient code by understanding how to use these operators and when to use them.

Here are some examples of arithmetic operators in Java and how they are used:

1. **Addition operator `+`:** This operator adds two operands together.
2. **Subtraction operator `-`:** This operator subtracts one operand from another.
3. **Multiplication operator `*`:** This operator is used to multiply two operands together.
4. **Division operator `/`:** This operator divides one operand into another.
5. **Modulo operator `%`:** This operator finds the remainder when one operand is divided by another.

InfoWarningTip
These operations are all supported for int, float, and double data types in Java.

```
int a = 20;
int b = 10;

int result;

result = a + b; // 30
result = a - b; // 10
result = a * b; // 200
result = a / b; // 2
result = a % b; // 0
```

#Comparison Operators

Comparison operators in java are used to compare two values. These operators are commonly used in conditional statements, such as `if` and `while` loops, to control a program's flow based on the comparison's outcome.

Here are some examples of comparison operators in Java and how they are used:

1. **Equal to operator `==`:** This operator checks if two operands are equal.
2. **Not equal to operator `!=`:** This operator is used to check if two operands are not equal.

3. **Greater than operator** `>`**:** This operator is used to check if the left operand is greater than the right operand.
4. **Less than operator** `<`**:** This operator is used to check if the left operand is less than the right operand.
5. **Greater than or equal to operator** `>=`**:** This operator is used to check if the left operand is greater than or equal to the right operand.
6. **Less than or equal to operator** `<=`**:** This operator is used to check if the left operand is less than or equal to the right operand.

They are supported for primitive data types, and the result of a comparison is a boolean value, true or false.

```
int a = 5;
int b = 2;

boolean result;

//Each of the following lines will re-set the value of result
according to the truth value of the statement. If the
statement evaluates to false, result will equal false. If it
evaluates to true, result will evaluate to true:
result = a == b; // result is now false, because 5 is not
equal to 2.
result = a != b; // result is now true because 5 is not equal
to 2.
result = a > b; // result is now true because 5 is greater
than 2
result = a < b; // result is now false because 5 is not less
than 2
result = a >= b; // result is now true because 5 is greater
than or equal to 2
result = a <= b; // result is now false because 5 is not less
than or equal to 2
```

#Logical Operators

Logical operators are used to perform logical “and,” “or,” and “not” operations. They combine two or more conditions/constraints or complement the evaluation of the original condition under particular consideration.

1. **The or Operator** `||` –> `a||b`**:** This operator returns **true** if **at least one of the operands is true**. Think of it as, "if a or b is true, the whole thing is true."
2. **The and Operator** `&&` –> `a&&b`: This operator returns **false if at least one of the operands is false**. Think of it as, "both a and b must be true to make this statement true. If one of them is false, the whole thing is false–it has to be 'and'".
3. **The not Operator** `!` –> `!a` **:** This unary operator returns the **reverse** of the operand. Rather than asking if something is equal to something else, you can be exclusionary with the `!` and say, "the following should happen as long as this variable is not a."

InfoWarningTip
Keep in mind that while using the `&&` operator, the second condition is not evaluated if the first one is false. Whereas while using the OR operator, the second condition is not evaluated if the first one is true. That is known as the short-circuiting effect.

```
boolean a = true;
boolean b = false;

boolean and = a && b; // "and" is false.
boolean or = a || b; // "or" is true.
boolean not = !a; // "not" is false.

int a = 10;
int b = 8;
int c = 12;

boolean and = (a > b) && (b > c); // "and" is false because (a
> b) is true and (b > c) is false.
boolean or = (a > b) || (b > c); // "or" is true because (a >
b) is true and (b > c) is false. Only one condition needs to
be met in order for this condition to be declared true.
boolean not = !(a > b); // "not" is false because (a > b) is
true
```

#Unary Operators

Java unary operators are the types that need only one operand to perform any operation like increment and decrement. Let's look at the various unary operators and see how they operate.

1. **Increment** `++`: It is used to increment the value of an integer. It can be used in two ways:
   a. Post-increment `a++`:
      When the `++` operator is placed after the operand, the value of the operand is incremented. Still, the previous value is retained temporarily until the execution of this statement, and it gets updated before the execution of the next statement.
   b. Pre-increment `++a`:
      When placed before the operand, the operand's value is incremented instantly.

2. **Decrement (--)**: It is used to decrement the value of an integer. It can be used in two ways:
   a. Post-decrement `a--`:
      When the `--` operator is placed after the operand, the value of the operand is decremented. Still, the previous value is retained temporarily until the execution of this statement, and it gets updated before the execution of the next statement.
   b. Pre-decrement `--a`:
      When placed before the operand, the operand's value is decremented instantly.

```
/* Post-Increment */
int a = 5;
```

```
int b = a++;

System.out.printf("a=%d, b=%d", a, b); // Prints out: a=6, b=5

/* Pre-Increment */
a = 5;
b = ++a;

System.out.printf("a=%d, b=%d", a, b); // Prints out: a=6, b=6

/* Post-Decrement */
int a = 5;
int b = a--;

System.out.printf("a=%d, b=%d", a, b); // Prints out: a=4, b=5

/* Pre-Decrement */
a = 5;
b = --a;

System.out.printf("a=%d, b=%d", a, b); // Prints out: a=4, b=4
```

InfoWarningTip
The post-increment and post-decrement `a++` or `a--` are more commonly used, but it's important to recognize that pre- and post- work in different ways. If your code is behaving oddly, check and see which one you've used.

#Knowledge Check

What is the value of z after the following code is run?

```
int x = 10;
int y = 5;
boolean z = x > y;
 // z =
```

What is the value of z after the following code is run?

```
int x = 10;
int y = 5;
int z = 12;
boolean w = ( x > y ) && ( y < z ) && !( x > z );
// w =
```

Which of the following conditions will resolve to a boolean value of `false`?

1 != 5

true && true

true && false

5 < 3

What will the value of a be at the end of this code snippet?

```
int a = 5;
a++;
a = a - 2;
// a = ?
```

6

5

4

3

```
int a = 10;
int b = 17;
int c = 12;

boolean or = (a > b) || (b > c);
```

In the code snippet above, both conditions would need to be false in order for the boolean "or" to be declared false. Therefore, the value of "or" is .

#Summary

- Arithmetic expressions include expressions of type int and double.
- The arithmetic operators consist of `+`, `-`, `*`, `/`, and `%` (modulo for the remainder in division).
- An arithmetic operation using two int values will evaluate an int value. With integer division, any decimal part in the result will be thrown away, essentially rounding down the answer to a whole number.
- An arithmetic operation that uses at least one double value will evaluate to a double value.
- Operators can be used to construct compound expressions.
- During the evaluation, operands are associated with operators according to **operator precedence** to determine how they are grouped. (`*`, `/`, `%` have precedence over `+` and `-` unless parentheses are used to group them.)
- An attempt to divide an integer by zero will result in an `ArithmeticException`.
- The assignment operator `=` allows a program to initialize or change the value stored in a variable. The value of the expression on the right is stored in the variable on the left.

- During execution, expressions are evaluated to produce a single value.
- The value of an expression has a type based on the evaluation of the expression.

#Extra Resources

Read more about the many operators available to the user on w3schools
Go through Josh Comeau's wonderful operator lookup!