Operators and Operands

by Sophia

WHAT'S COVERED

In this lesson, you will learn about how to use operators to perform mathematical operations on integers and float variables. Specifically, this lesson covers:

1. Common Mathematical Operators

Variables can be used to contain data in memory, as discussed previously. However, in order to make variables and literal values useful, mathematical operations are required.

An expression is useful when working with values, such as variables, literal values, and mathematical operators. In an earlier tutorial, you learned that an expression is a piece of Java code that combines values and operators to produce a value in the program. An expression always produces a definite value when it is evaluated.

The following arithmetic operators provide the basis for carrying out calculations in Java.

Arithmetic Operation	Java Operator	Sample Java Expression
Addition	+	int sum = 2 + 3;
Subtraction	-	float difference = 7.25 - 2.3;
Multiplication	*	double circumference = 5.5 * 3.14159;
Division	/	float quotient = 9.0f / 5;
Modulus	%	int remainder = 23 % 8;

The plus sign for addition and minus sign for subtraction should already be familiar as they relate to adding and subtracting numbers. An asterisk is used as the multiplication operator. It's important to note that there is no

multiplication sign resembling the letter ‘x’ in Java expressions. The forward slash can be used as the division operator. It can be seen when writing fractions like 1/2 and 2/3. The modulus operator will be addressed below.

When working with integers in Java, it is important to be aware of integer division. When a value or variable with the data type `int` is divided by another `int`, the resulting quotient is always an integer. The decimal that might otherwise be expected is simply dropped.

TRY IT

Directions: Type the following code into a file named IntigerDivision.java on the IDE:

```
public class IntegerDivision {
  public static void main(String[] args) {
        int two = 2;
        int three = 3;
        int four = 4;

        System.out.println("Integer Division Examples: ");

        int integerQuotient = three / two;
        // The + is for concatenation not addition
        System.out.println("\t3 / 2 = " + integerQuotient);

        integerQuotient = four / two;
        System.out.println("\t4 / 2 = " + integerQuotient);

        integerQuotient = two / three;
        System.out.println("\t2 / 3 = " + integerQuotient);
  }
}
```

Next, run the program.

The IntegerDivisinon.java program output should look like this:

```
Integer Division Examples:
    3 / 2 = 1
    4 / 2 = 2
    2 / 3 = 0
```

REFLECT

You can think of integer division somewhat as “all or nothing.” Either you get a whole number out or nothing (0) if the result is less than 1. And speaking of 0, Java does one other thing when performing integer division. If you

divide by integer 0, you will get a run-time error. We'll be discussing how to handle errors like this later but for now just know that your program will stop running if you try to divide by integer 0.

---

2. Modulus Operator

The discussion of integer division is a good place to introduce the **modulus operator (%)**. While integer division produces a whole-number quotient without retaining the remainder, the modulus operator carries out integer division; however, instead of returning the quotient, it returns the remainder.

The following code illustrates the functioning of the modulus operator:

```
public class Modulus {
  public static void main(String[] args) {
    int two = 2;
    int three = 3;
    int four = 4;

    System.out.println("Modulus Examples: ");

    int modulus = three % two;
    // The + is for concatenation not addition
    System.out.println("\t3 % 2 = " + modulus);

    modulus = four % two;
    System.out.println("\t4 % 2 = " + modulus);

    modulus = two % three;
    System.out.println("\t2 % 3 = " + modulus);
  }
}
```

Running this code in a file called Modulus.java produces the following results:

```
Modulus Examples:
    3 % 2 = 1
    4 % 2 = 0
    2 % 3 = 2
```

The modulus operator can be very useful.

EXAMPLE It can determine whether one number is divisible by another:

> if x % y is zero, then x is divisible by y.

Using the example above, it can be discovered that 4 is divisible by 2. Additionally, it can confirm that 4 is a multiple of 2, and thus 4 is an even number.

EXAMPLE The right-most digit or digits can be extracted from a number:

> x % 10 yields the right-most digit of x (in base 10).

Similarly, x % 100 yields the last two digits.

TRY IT

Directions: Type the following code into a file named ModulusPlaceValue.java in the IDE:

```
public class ModulusPlaceValue {
  public static void main(String[] args) {
    int number = 6849;

    System.out.println("Using modulus to get rightmost digit(s): ");
    // Get the rightmost digit using % 10
    int modulus = number % 10;
    System.out.print("The rightmost digit in " + number);
    System.out.print(" is " + modulus + " (" + number);
    System.out.println(" % 10).");
    // Get the rightmost 2 digits using % 100
    modulus = number % 100;
    System.out.print("The rightmost 2 digits in " + number);
    System.out.print(" are " + modulus + " (" + number);
    System.out.println(" % 100).");
  }
}
```

The output screen should show the following results:

```
Using modulus to get rightmost digit(s):
The rightmost digit in 6849 is 9 (6849 % 10).
The rightmost 2 digits in 6849 are 49 (6849 % 100).
```

REFLECT

The output should show 9 and 49 as right-most digits.

TERM TO KNOW

Modulus Operator (%)

The modulus operator returns the remainder of the two numbers after division. It is represented in Java by the (%) symbol.

3. Increment and Decrement Operators

When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence. When more than one operator appears in an expression, they are evaluated according to order of operations.

Java generally evaluates operators from left to right. However, when working with the following numeric operators, Java follows mathematical conventions:

- Multiplication and division take “precedence” over addition and subtraction. This means that they happen first. So, 1 + 2 * 3 yields 7, not 9. And, 2 + 4 / 2 yields 4, not 3.
- If the operators have the same precedence, they are evaluated from left to right. In the expression minute * 100 / 60, the multiplication happens first. If the value of minute is 59, then 5900 / 60 yields 98. If these same operations had gone from right to left, the result would have been 59 * 1, which is incorrect.
- Any time you want to override the order of operations, or you are not sure what the order is, you can use parentheses. Expressions in parentheses are evaluated first, so (1 + 2) * 3 is 9. You can also use parentheses to make an expression easier to read. For example, (minute * 100) / 60. This is accurate even though it doesn’t change the result.

HINT

Do not spend a lot of time trying to remember the order of operations, especially for other operators. If it is not obvious what the order of operations are by looking at the expression, use parentheses to make it clear.

IN CONTEXT

Review the scenarios below to see how order of operations becomes important.

Scenario 1

Let’s say there are 20 pizza slices. You have eaten two and then gave half of the remaining slices to your colleagues at work. How many pizza slices are left? How should the calculation be structured? Consider the following piece of code (note: it is not a complete program):

```
int pizzaSlices = 20;
int remainingSlices = ; // Fill in the expression
System.out.println(remainingSlices);
```

```
int pizzaSlices = 20;
int remainingSlices = (pizzaSlices - 2) / 2;
System.out.println(remainingSlices);
```

Scenario 2

Consider that you've invested $300.00 in a new company. The company had to use a third of the initial funds for marketing. They then used $50 for product development. They were able to successfully deploy the project. As a return for investment, they gave 5 times the remaining funds to each investor. How much money was returned to you?

```
float investment = 300.00f; // f indicates value is a float rather than double
float money = ;
System.println(money);
```

```
float investment = 300.00f;
float money = ((investment * 2 / 3) - 50) * 5;
System.out.println(money);
```

How much profit did you make from that initial investment?

```
float investment = 300.00f;
float profit = ;
System.out.println(profit)
```

```
float investment = 300.00f;
float profit = ((investment * 2 / 3) - 50) * 5 - investment;
System.out.println(profit);
```

4. Compound Assignment Operators

In addition to the simple assignment operator (=), Java supplies a number of shortcut assignment operators. These shortcuts allow users to combine an arithmetic operation and an assignment in one operation. These operations can be used with either integer or floating-point operands.

The `+=` operator allows you to combine addition and assignment into one expression.

The statement:

```
count += 3;
```

is equivalent to:

```
count = count + 3;
```

The following is a list of Java's assignment operators:

Operator	Operation	Example	Interpretation
=	Simple assignment	m = 3;	m = 3;
+=	Addition then assignment	m += 3;	m = m + 3;
-=	Subtraction then assignment	m -= 3;	m = m - 3;
*=	Multiplication then assignment	m *= 3;	m = m * 3;
/=	Division then assignment	m /= 3;	m = m / 3;
%=	Remainder then assignment	m %= 3;	m = m % 3;

SUMMARY

In this lesson, you learned about the **common mathematical operators** that can be used in Java to perform mathematical operations on integers and float variables. You also learned how the **modulus operator** works. You explored **increment and decrement operators** to learn why the order of operations is extremely important. Finally, you used **compound assignment operators** to combine an arithmetic operation and an assignment in one operation that follows the conventional mathematical order of operations.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from "Python for Everybody" By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Modulus Operator (%)

The modulus operator returns the remainder of the two numbers after division. It is represented in Java by the (%) symbol.