Mathematical Operations in Java

Basic Mathematical Operations

Many mathematical expressions in Java look identical to those that you would write on paper. There are five basic arithmetic operators in Java, as shown below.

Java Operator	Purpose	Example
+	Addition	5 + 3.4 = 8.4
-	Subtraction	203.9 – 9.12 = 194.78
*	Multiplication	3 * 4.7 = 14.1
/	Division	17.889 / 6.7 = 2.67
%	Modulo Division (calculates the remainder)	35 % 9 = 8

In general, operations on values of the same type will produce an answer that is also of the same type. It is particularly important to keep this in mind when performing an integer division. Consider the following examples:

(a) 15 / 3 gives 5

(b) 13 / 4 gives 3 (rather than 3.25)

(c) 9 / 5 gives 1 (rather than 1.8... notice that the answer is *truncated*, not rounded)

Whenever integer and floating point values are mixed in an expression, the result will be a floating point value. Notice the subtle difference this creates when applied to our previous example.

(a) 15 / 3.0 gives 5.0 (a floating point answer)

(b) 13.0 / 4 gives 3.25

(c) 9 / 5.0 gives 1.8

Modulo Division

It is worth giving an extra moment to consider and understand the *modulo operator*. This operator is used to find the *remainder* from an *integer division*, which is actually the first type of division taught to children in elementary school.

Consider the example of 35 % 9 = 8. To begin, we ask, “How many times does 9 divide evenly into 35?” Simply following the nine-times-table yields

$$9\times1=9 \qquad 9\times2=18 \qquad 9\times3=27 \qquad 9\times4=36$$

The answer is obviously 3 times. Now, what is the remainder? Subtract 27 from 35 for a remainder of 8.

This mathematical concept is actually quite useful and frequently used in programming, which is why many programming languages, including Java, provide it in a compact and convenient form.

Division by Zero

Both regular division and modulo division pose the risk of accidentally performing an illegal mathematical operation, which is *division by zero*. In mathematics, division by zero is *undefined*, and this must apply to computer programs as well.

Since most calculations in programs use *variables*, it may not be obvious when or if this is going to happen. It is always a good idea to carefully consider the possible values of any variables when you perform a division operation.

If a division by zero occurs while performing an integer operation, the Java program will exit with an error. This is known as **throwing an exception**, and in this case, the exception is an **ArithmeticException**. Basically, the program is informing the user that it has encountered a situation that it doesn't know how to handle. Later we will learn about handling various types of exceptions, but for now, we will simply try to avoid this situation.

When performing floating point operations, the division by zero is handled by Java using one of three special floating point expressions, as summarized in the table below.

Floating Point Operation	Result	Can be used in further calculations?	Will print as the string...
positive value divided by zero	positive infinity	yes	`Infinity`
negative value divided by zero	negative infinity	yes	`-Infinity`
zero divided by zero	not a number	no	`NaN`

Mathematical Operations in Java

Mathematical Methods

To allow for mathematical operations beyond basic arithmetic, Java provides additional functionality using a large group of *methods* contained in the *Math* class. To use any of these methods, we must identify them as being part of the Math class. Some of the more common and useful methods from the Math class are shown in the following table.

Mathematical Operation	Java Syntax	Description
$\sqrt{(x)}$	`Math.sqrt(x)`	square root of x
x^y	`Math.pow(x, y)`	x raised to the exponent y
$\lvert(x)\rvert$	`Math.abs(x)`	absolute value of x

Some other important mathematical methods do not correspond to traditional mathematical operations, but are very useful in programming.

Java Syntax	Description
`Math.round(x)`	rounds a floating point value, x, to the nearest integer (up or down)
`Math.ceil(x)`	rounds a floating point value up to an integer value
`Math.floor(x)`	rounds a floating point value down to an integer value
`Math.random`	generates a random number between 0 and 1 ($0 \le x < 1$) includes the value 0, but not the value 1 (0 to 0.999999...)

Exercises

1. A quadratic equation of the form $a x^2 + b x + c = 0$ has roots (zeroes)

   $$\frac{-b \pm \sqrt{b^2 - 4 a c}}{2 a}$$

   Write a fragment of code that *efficiently* determines the values of the roots (root1 and root2) of the quadratic equation with coefficients a, b, and c. Assume that all the variables have been declared as type `double` and that the equation has two real roots.

2. Each of the following expressions is intended to evaluate the expression

$$\frac{a\,x^2+b}{c\,x+d}$$

Some are correct while others are not. Classify each as being correct or incorrect. For those that are incorrect, give the reason(s).

(a) `( a * x * x + b ) / ( ( c * x + d )`

(b) `( a * Math.pow(x, 2) + b ) / ( c * x + d )`

(c) `( (a) (x) (x) + (b) ) / ( (c) (x) + d )`

(d) `( b + x * ( x * (a) ) ) / ( d + x * (c) )`

(e) `(a) * Math.pow(x,2) + (b) / ( (c) * (x) + d )`

(f) `( a * ( x * x ) ) + (b) / c * (x) + d`

3. Suppose that the following declarations have been made.

```
int i = 3, j = 4, k = 2;
```

Using these starting values in each part, find the value of each variable after the given statement has been executed.

(a) `j = ++i * k--;`

(b) `i = --j + k/2;`

(c) `k = i-- - j++;`

(d) `j = (2*i++)%(++k + 1);`

(e) `i += j -= --k;`

(f) `i *= j /= k++;`

4. Write a program that asks the user for a three-digit number, finds the sum of the digits of the number, and then prints both the number and its digit sum.