Arithmetic in Java

- Operators: Addition (+), subtraction (-), multiplication (\*), division (/).

- For floating point numbers, division will work as expected. But for integers, we will go back to our very first experiences with division because it deals with *whole number* division only.

- Example: In Java, a division calculation could compute: 5/2 = 2. Just that whole number part. Whenever we divide two integers, we must remember this.

- IMPORTANT example:

*Double ans = 2/4;*

*// ans is now 0.0*

This answer only applies to INTEGERS.

*ans = 2.0/4;*

*// ans is now 0.5*

Integer division will only apply if both operands are integers. As long as one of the operands is a floating point number, we’ll get floating point division.

I have two int variables: total and count. I also have a double variable named average. If total has the value 35 and count has the value 4, what will be the value stored in average after this line of code: average = total / count;

A. 8.75

B. 8

C. 9

Explanation: Both variables total and count are int variables. So when we divide two int variables, we will only compute how many times the divisor (4) fits into the dividend (35).

- The solution to this problem is casting (or typecasting.) This doesn’t change the original value of the int variable. It makes a copy of the value in the new type and then we use the copy. Here is the syntax:

*average = (double) total /count;*

- The cast converts the int value of total into a double type value, and the division is then floating point division.

- Modulus: remainder operator. It only applies to integer values and it gives the remainder after the division.

“Five mod two” = 5 % 2 = 1 because 2 goes into 5 twice and the remainder is 1.

\*For positive numbers, the modulus and the remainder are always the same.

4 % 10 = 4 because 4 goes into 10 zero times and 4 is left over, indivisible.

- Operator precedence:

1) Parentheses, inside out.

2) Typecasting.

3) \*, /, % (left to right)

4) +, - (left to right)

- Casting can be our greatest challenge with precedence and arithmetic.

Example:

*int num1 = 2, num2= 3, num3 = 4;*

*double result;*

*result = (double) num1 + num2 / num3;*

The problem here is that we make num1 a double, but we still apply integer division to num2 and num3 before the (double) gets involved.

So the result it 2.0, not the desired 2.75:

*num2/num3 = 0*

*(double) num1 + 0 = 2.0*

- One way people try to fix this is to use parentheses, but we’ve really just said do the whole calculation and then convert it to a double.

*Result = (double) (num1 + num2 / num3);*

- That’s going to convert it to a double right before we assign it into a double. But the calculations(both division and addition) are done with integers. So we end up with 2.0 again.

- The correct way to deal with this is to actually put the cast directly onto either num2 or num 3.

*Result = num1 + (double) num2 / num3;*

- So we can put it on num2 or num 3. If we do that, we will get the floating point division we were looking for and the result of 2.75.

- Another mistake that is commonly made is putting parentheses around the entire dividend and divisor outside of the double casting, like so:

*result = num 1 + (double) (num2 / num3)*

- Which just computes to 2.0 once again. It divides before the cast.