In this problem set we will investigate how java preforms it’s arithmetic calculations. The idea behind this problem set is to ensure we know what Java does behind the scenes when it is performing it’s calculations.

For each of the examples below you can replace the output statement with the appropriate expression as shown below: (or use the Interactive Pane)

class ProblemSet27 {

public static void main(String [] arg) {

// you will change the expression below with the one you are testing in the table

System.out.println(“The value is: “ + ***(<expression your testing goes here>)*** );

} // end of main

} // end of ProblemSet5

|  |  |
| --- | --- |
| **Expression**  **Tested** | **State the value and the type of operation – Integer or Floating Point(decimal). Explain surprises if any.** |
| (5 + 3) | 8 - no surprise, straight integer addition |
| ( 17 / 5 ) | 3- 17 devided by 5 does not normally equal 3 |
| ( -17 / 5) | -3 – still confused on how but it makes sense if the last one was 3 |
| ( 17/ -5 ) | -3 – Same thing again |
| ( -17/ -5) | 3- no surprise |
| (17.0 / 5) | 3.4 – I see how this works now, because it has no decimals to work with before it rounded, now it has decimals |
| (-17.0 / 5) | -3.4 |
| (17 / 5.0) | 3.4 |
| (17 / -5.0) | -3.4 |
| (-17/ -5.0) | 3.4 |
| **Expression**  **Tested** | **State the value and the type of operation – Integer or Floating Point(decimal). Explain surprises if any.** |
| ((2+5) / 4) | 1. Should be 2, so it rounds down not up |
| (0.5 + 0.5) | **1.0** |
| (1.0/2 + 1.0/2) | 1.0, make sense |
| (1/2 + 1/2) | **0 – makes sense cause it rounds down** |
| (12.0 \* 30) | **360.0 – no surprises** |
| (12.0 \* 31) / 12 | **31.0 – no surprise** |
| ( 12 % 4 ) | 1. **No surprises since % returns the remainder of a division equation. 12/3** |
| ( 12 % 5 ) | **2 – Yup returns remainder** |
| ( -12 % 5) | **-2 - - / + = -** |
| ( -12 % -5) | **2 – - / - = +** |
| (12 % -5) | **-2 - +/- = -** |
| (12 % -4) | **0 cause no remainder** |
| ((5 + 0.0)/-4) | **-1.25 – 5/4 is 1.25, adding 0 just let it do decimals** |
| ((5 + 1)/-4.0) | **-1.5 – No clue why its only 1 decimal space** |
| ((1/2+3.5)/3.5) | **-1.0, not a clue why** |
| ((1.0/2+3.5)/3.5) | **1.1428571428571428, that’s a lot of decimals** |
| ((1/2.0 + 3.5)/3.5) | **-1.1428571428571428- same thing as before** |
| (34-30)\*(20-10) | **40 – nothing surpriseing, just 4 \* 10** |
| (34.0 – 30)\*(20-10) | **40.0** |
| (34 – 30) \* (20.0 – 10) | **40.0** |
| **Expression**  **Tested** | **State the Value and type of operation – Integer or Floating Point(decimal). Explain surprises if any.** |
| (2400000000000 / 8000000) | **I cant run it, numbers too big** |
| (2400000000000000.0 / 8000000) | **3.0E8, numbers to big so it gives us a hex code** |
| (3F / 4) | **(HEX CODES ARE YOU SERIOUS. I had to get my chart from last year for this)** |
| (3D / 4) | **0.75 – I gotta do Hex math to double check** |
| (3 / 4F) | **0.75- Same thing, I hate Hex conversions, its rounding** |
| (3 / 4D) | **0.75 – It keeps rounding to 0.75** |
| (3D / 4D) | **0.75 – Keeps rounding** |
| (3F / 4F) | **0.75 – Again rounding** |

What do you think the letters stand for in the last 5 examples above?

Hexidecimal. I don’t like this, I’d rather do binary, atleast I remember that. (I had to write my name in Hex last year and I don’t remember what I did)

Hint: Try assigning them to variables like below:

int x = (3 / 4D);  
int y = (3/ 4F);

How do you fix the errors that these two lines of codes generate?

**Definition**

**Expressions**

An **expression** is a combination of literals, operators, variable names, and parentheses used to calculate a value.

Expressions contain *operators* and *operands*. You already know what an operator is (a symbol such as +, -, \*, or / that calls for an arithmetic operation).

An **operand** is a value that is acted upon by an operator.

The parts of an expression must be arranged correctly. The syntax of Java describes the correct arrangements. The rules for correct Java expressions are about the same as for algebra. Essentially,

1. Each operator must have the correct number of operands.
   * Multiplication \*, Division /, Addition +, Subtraction - should have two operands, one on each side.
   * Negation - and unary plus + should be followed by one operand.
2. Parentheses () can surround a legal expression to make it an operand.

In Java expressions, operators and operands must be explicit. In 13 - 5 the 13 and the 5 are the operands and the - is the operator.